НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)
Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Корпоративний бот для IT-компаній*

Виконав: студент IV курсу, групи *ІП-63 Васюк В. В.* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст. в., Коротенко А.А.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

ст. в., Алещенко О.В.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

ID перевірки:
1003947605

Дата перевірки:
11.06.2020 00:54:30 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
11.06.2020 22:17:47 EEST

ID користувача:
77149

Назва документу: Vasjuk_ip63

ID файлу: 1003961320 **Кількість сторінок:** 60 **Кількість слів:** 9947 **Кількість символів:** 73761 **Розмір файлу:** 84.56 KB

2.16% Схожість

Найбільша схожість: 1.53% з джерело https://ela.kpi.ua/bitstream/123456789/31004/1/Lysenko_bakalavr.pdf

1.57% Схожість з Інтернет джерелами	4	Page 62
2.16% Текстові збіги по Бібліотеці акаунту	48	Page 62

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Не знайдено замінених символів

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Васюку Владиславу Валерійовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Корпоративний бот для ІТ-компаній*»

керівник проєкту *Коротенко Артем Андрійович, ст. в.*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: загальні положення, змістовний опис і аналіз предметної області, аналіз успішних ІТ-проєктів, аналіз вимог до програмного забезпечення

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних

3) Аналіз якості та тестування програмного забезпечення: вступ та призначення плану тестування, предмети тестування, функціонал що підлягає тестуванню, функціонал що не підлягає тестуванню, підхід до тестування, критерії проходження тестування, критерії припинення тестування, вимоги до

середовища, опис тестів

4) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення, робота з програмним забезпеченням

5. Перелік графічного матеріалу

1) Схема структурна бізнес процесів

2) Схема структурна розгортання

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*15.03.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*22.03.2020*	
3.	*Постановка та формалізація задачі*	*25.03.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*29.03.2020*	
5.	*Алгоритмізація задачі*	*05.04.2020*	
6.	*Моделювання програмного забезпечення*	*12.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*15.04.2020*	
8.	*Розробка архітектури програмного забезпечення*	*20.04.2020*	
9.	*Розробка програмного забезпечення*	*30.04.2020*	
10.	*Налагодження програми*	*10.05.2020*	
11.	*Виконання графічних документів*	*17.05.2020*	
12.	*Оформлення пояснювальної записки*	*24.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*		
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Владислав ВАСЮК
(підпис)

Керівник _____ Артем КОРОТЕНКО
(підпис)

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6305.045490.01.81	Пояснювальна записка	70	
3	А4	КПІ.ІП-6305.045490.02.91	Технічне завдання	11	
4	А4	КПІ.ІП-6305.045490.03.13	Опис програми	19	
5	А4	КПІ.ІП-6305.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-6305.045490.05.34	Керівництво користувача	9	
7	А4	КПІ.ІП-6305.045490.06.33	Керівництво програміста	6	
8	А3	КПІ.ІП-6305.045490.07.99.СС	Схема структурна бізнес процесів	1	
9	А3	КПІ.ІП-6305.045490.08.99.СС	Схема структурна розгортання	1	
10	А3	КПІ.ІП-6305.045490.09.99.КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6305.045490.00.90		
Розробн.		Васюк В.В.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Коротенко А.А.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-63	
Н/контр.		Ліщук К.І.					
В.о.зав.каф		Павлов О.А.					

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 32 таблиці, 13 рисунків, 20 джерел – загалом 138 сторінок.

Об'єкт дослідження: автоматизація рутинних робочих процесів в ІТ-компаніях.

Мета дипломного проєкту: розробка модульного корпоративного бота для автоматизації рутинних процесів в ІТ-компаніях, а саме процесів оформлення відпусток і проведення опитувань серед співробітників, з можливістю для розробників додавати нові модулі.

У першому розділі було проаналізовано предметну область, доступні технічні рішення та аналоги. Описано варіанти використання, функціональні та нефункціональні вимоги.

У другому розділі було проаналізовано бізнес-процеси, спроєктовано архітектуру програмного забезпечення, спроєктовано структуру баз даних та обрано СКБД. Обрано мову програмування та інструменти для розробки, проаналізовано безпеку даних та впроваджено механізми для захисту інформації.

У третьому розділі описано план тестування, за яким проведено аналіз якості програмного забезпечення.

У четвертому розділі описано розгортання серверної та клієнтської частин додатку, а також наведено інструкції користувача та програмування нових модулів.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, ІТ-КОМПАНІЇ, ЧАТ-БОТ, SLACK, МІКРОСЕРВІСИ

ABSTRACT

Explanatory note of the diploma project consists of 4 sections, 32 tables, 13 figures, 20 sources – total 138 pages.

The object of study: automation of routine work processes in IT companies.

The aim of the diploma project: development of a modular business bot for automation of routine processes in IT companies, namely the processes of requesting a vacation and conducting surveys among employees, with the ability for developers to add new modules.

In the first section, the domain, available technical solutions and alternatives were analyzed. Use-cases, functional and non-functional requirements were described.

In the second section, business processes were analyzed, software architecture was designed, database structure was designed and DBMS was selected. The programming language and development tools have been selected, data security has been analyzed and data security mechanisms have been implemented.

The third section describes the test plan, according to which quality of the software was analyzed.

The fourth section describes the deployment of server- and client-sides of the application, as well as user instructions and instructions for programming new modules.

KEYWORDS: AUTOMATION, IT-COMPANIES, CHATBOT, SLACK, MICROSERVICES

Пояснювальна записка
до дипломного проєкту

на тему: «Корпоративний бот для IT-компаній»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HR – Human Resources – термін, яким в IT-компаніях зазвичай позначають спеціалістів в галузі управління персоналом.

API – Application Programming Interface – набір чітко визначених методів для взаємодії різних компонентів ПЗ.

JSON – JavaScript Object Notation – текстовий формат серіалізації даних.

REST – Representational State Transfer – підхід до архітектури мережевих додатків та API.

HTTP – Hypertext Transfer Protocol – протокол прикладного рівня для передачі веб-сторінок та інших даних через мережу.

TLS – Transport Layer Security – криптографічний протокол для захищеної передачі даних через мережу. Використання TLS разом з HTTP зазвичай носить назву **HTTPS** (Hypertext Transfer Protocol Secure).

HMAC – Hash-based Message Authentication Code – криптографічний метод перевірки цілісності інформації.

MIT (ліцензія) – група ліцензій для розповсюдження програмного забезпечення з відкритим програмним кодом.

Heartbeat – періодичний сигнал, що генерується апаратним чи програмним забезпеченням для індикації нормальної роботи.

ASGI – Asynchronous Server Gateway Interface – асинхронний протокол передачі запитів від веб-сервера до програм на мові Python.

WSGI – Web Server Gateway Interface – синхронний протокол передачі запитів від веб-сервера до програм на мові Python.

NoSQL – тип баз даних, що відрізняється від SQL-баз даних переважно відсутністю реляційних зв’язків.

BSON – Binary JSON – бінарний формат серіалізації даних, що використовується в MongoDB для збереження документів.

Lint (**linter**) – утиліти для статичного аналізу коду, зазвичай можуть знаходити помилки програмування, невідповідності стилю коду тощо.

TDD – Test-Driven Development – підхід до розобки програмного забезпечення, при якому розробка починається з попереднього написання тестів, після чого розробляється код, який має пройти ці тести.

Code coverage – міра, що позначає відсоток протестованого вихідного коду програми.

ВСТУП

Ефективна робота будь-яких ІТ-компаній потребує вирішення організаційних питань (оформлення відпусток і лікарняних, онбординг працівників, перегляд заробітних плат). В малих та середніх за розміром компаніях, ці рутинні задачі часто виконуються вручну менеджерами та співробітниками відділу кадрів, що значно витрачає їх час.

Для автоматизації цих процесів в ІТ-компаніях використовують різні програмні додатки чи сервіси. Серед таких сервісів варто виділити корпоративних ботів на базі месенджерів, наприклад Slack, Telegram і Microsoft Teams.

На фоні розвитку пандемії COVID-19 та переходу ІТ-компаній на віддалений режим роботи, такі месенджери використовуються як один з основних засобів зв’язку, через що зростає зручність використання для автоматизації саме ботів на базі цих платформ. Але більшість таких рішень розповсюджуються за платною щомісячною підпискою, що значно затруднює використання в малих та середніх ІТ-компаніях, через що в них продовжують виконувати рутинні процеси вручну.

Оскільки такі боти мають закритий програмний код, додатково з’являється проблема конфіденційності інформації – немає змоги побачити, як саме зберігаються, обробляються і використовуються дані, зібрані при користуванні. Також, через закритість коду немає можливості розширити функціонал (наприклад, додати нові команди чи інтеграції з іншими сервісами, що використовуються в компанії).

Метою написання даного дипломного проєкту є автоматизація рутинних процесів в ІТ-компаніях.

Завданням даної дипломної роботи є створення модульного корпоративного бота для Slack з відкритим програмним кодом, з можливістю зручно розширювати функціонал шляхом додавання плагінів (модулів). Також в рамках роботи буде реалізовано модуль для автоматизації затвердження

відпусток чи відгулів, і модуль створення опитувальників для збору інформації серед працівників. Компонентами готового ПЗ є головний сервер для взаємодії між модулями і Slack API, бібліотека для написання модулей (плагінів), модуль відпусток та модуль опитувань.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Ринок ІТ в Україні протягом останніх років стрімко розвивається. Кількість робітників в цій сфері постійно зростає; формуються як нові продуктові компанії, так і аутсорсингові, продовжують рости існуючі компанії [1]. Незважаючи на це, проблема автоматизації робочих процесів в даній індустрії досі не має однозначного рішення, адже досі не існує інструмента, який би міг підійти будь-якій компанії для всіх задач.

В даному контексті, **автоматизація** полягає у впровадженні програмного забезпечення чи сервісу для виконання повторюваних задач без втручання людини. В роботі ІТ-компаній є безліч можливих рутинних задач для автоматизації – ознайомлення нових співробітників, оформлення лікарняних, відпусток, проведення планових переглядів заробітної плати. В малих ІТ-компаніях дуже часто такі задачі вирішують покласти на плечі відділу HR або менеджерів, що витрачає робочий час цих співробітників якщо вони виконують це вручну.

Автоматизація значно підвищує ефективність і продуктивність працівників ІТ-компаній, зменшує витрачений робочий час і може призвести до збільшення прибутків в цілому [2].

1.2 Змістовний опис і аналіз предметної області

Розглянемо ситуації і запити працівників, які часто виникають у процесі роботи невеликих ІТ-компаній. Припустимо, що компанія не використовує платні сервіси або системи для автоматизації, а всі обов’язки покладено на відділ HR чи інших працівників і виконуються вручну.

Ведення обліку відпусток чи лікарняних. Якщо ведення обліку виконується вручну, з часом це може призвести до наступних проблем.

– **Визначення кількості доступних днів у конкретного працівника**: потрібно вручну переглядати всю історію відпусток чи лікарняних працівника за рік, або будувати складні формули в Excel для підрахунку. Це може значно витратити час відділу чи людини, яка відповідає за облік. Автоматизація надасть можливість працівникам самостійно дізнатися, на скільки днів можна піти у відпустку, та зекономить час.

– **Оголошення співробітників, що планують або вже вийшли у відпустку**: якщо працівник забув або не зміг попередити свою команду перед виходом у відпустку, це може неочікувано порушити плани. Автоматизація обліку дозволить, наприклад, автоматично розсилати сповіщення працівникам, коли хтось з їхньої команди планує вийти у відпустку чи захворів.

– **Помилки при веденні обліку**: при веденні обліку в Excel, відповідальна людина може допустити помилку, наприклад, при введенні дат початку та кінця відпустки. Якщо процес автоматизовано, такі помилки в більшості випадків відловляться ще перед внесенням даних в систему.

Проведення опитувань. В компаніях часто виникає потреба організувати опитування серед працівників, наприклад під час перегляду заробітньої плати (напр. «Чи можете ви покластися на цього працівника під час роботи?», «Чи виявляє цей працівник ініціативу щодо вирішення проблем?» тощо), або після проведення зустрічі (напр. «Чи сподобався вам такий формат зустрічі?», «Як можна покращити цю зустріч наступного разу?»). Також, це можуть бути опитування пов’язані з позаробочими процесами – вибір місця проведення корпоративу, вибір страв на обід, подарунка на день народження колеги тощо.

В багатьох випадках такі опитування проводяться вручну через електронну пошту або за допомогою сервісу Google Forms, і потрібно витрачати додатковий час на нагадування працівникам котрі не надали відповіді. Автоматизація цього процесу дозволить мінімізувати час, витрачений як на створення і відправку опитувань кожному з учасників, так і на збір відповідей.

Як видно з наведених прикладів, автоматизація може значно скоротити час працівників та підвищити ефективність роботи компанії.

В рамках дипломної роботи було обрано наступні задачі для автоматизації, оскільки в певній мірі вони зустрічаються в будь-якій компанії. Інші задачі розробники зможуть реалізувати самостійно шляхом створення нових модулів, зважаючи на потреби конкретної компанії.

1.2.1 Опис процесу автоматизації оформлення відпусток чи відгулів

Даний процес може значно відрізнятися в залежності від законодавства країни, в якій знаходиться IT-компанія, від культури компанії та інших особливостей. Через це неможливо в повній мірі описати його таким чином, щоб повністю охопити всю діяльність. В загальному вигляді, процес зображено у вигляді схеми структурної на рисунку 1.1.

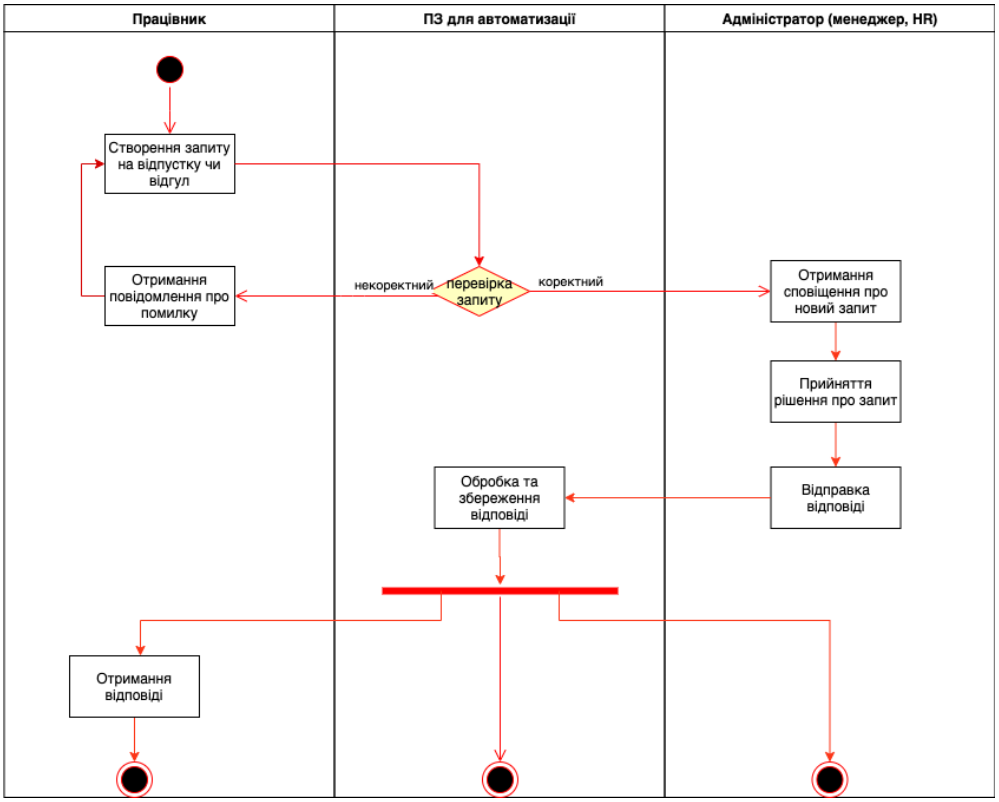


Рисунок 1.1 – Схема структурна діяльності процесу оформлення відпусток

Спочатку працівник компанії, який хоче піти у відпустку чи взяти відгул, створює запит, в якому вказує бажані дати, тип та причину. Після цього запит проходить перевірку – наприклад, що дати введено вірно. Якщо запит проходить перевірку, він відправляється до співробітників-адміністраторів системи (це можуть бути лідер команди, менеджер, HR тощо), які приймають рішення. Остаточне рішення відправляється працівнику, а також може зберігатися в базі даних чи інших сервісах в історичних цілях.

1.2.2 Опис процесу автоматизації проведення опитувань

Схему структурну діяльності цього процесу зображено на рисунку 1.2.

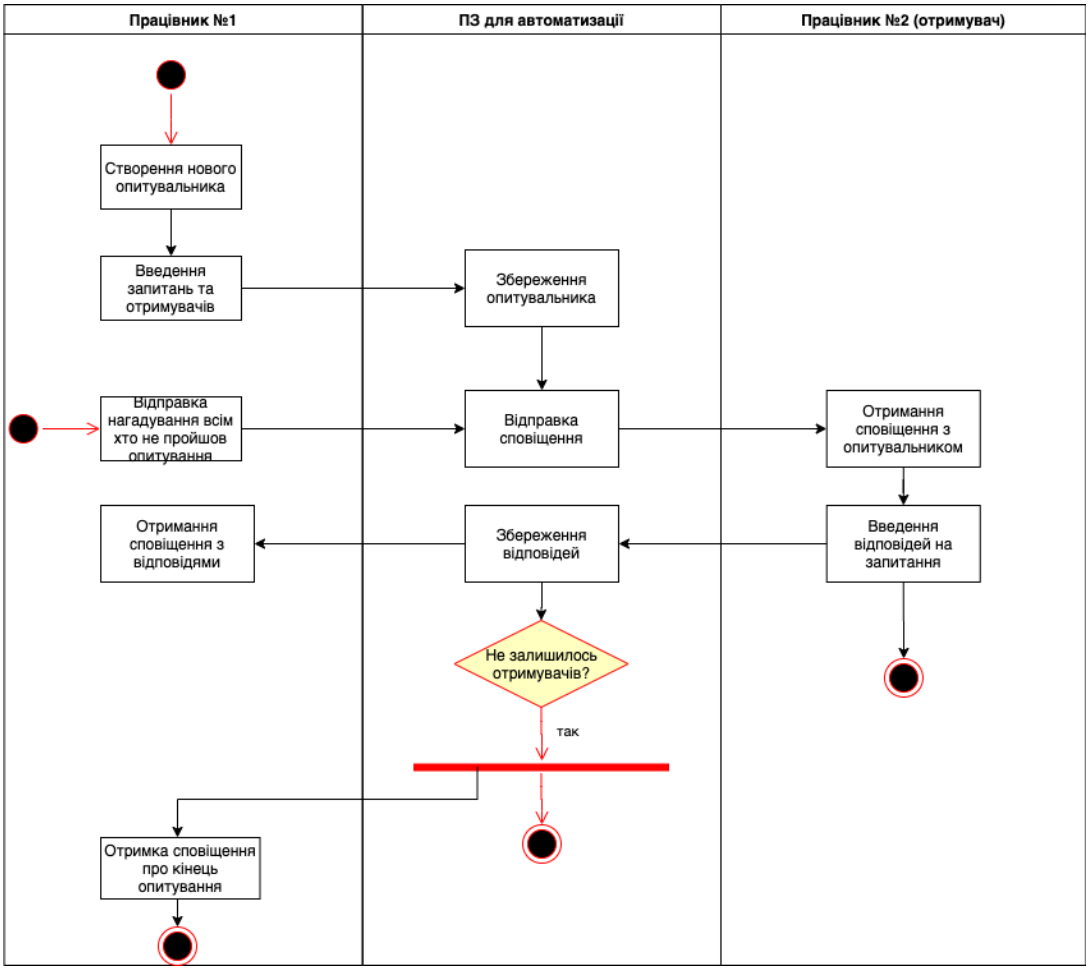


Рисунок 1.2 – Схема структурна діяльності процесу проведення опитування

Спочатку працівник, що хоче провести опитування, вводить запитання та обирає отримувачів. Ці дані зберігаються в базі даних та розсилаються всім учасникам. В разі потреби, автор опитувальника може відправити сповіщення повторно, щоб нагадати учасникам які забули надати відповіді. Учасники заповнюють опитувальник та відправляють відповіді, а після того як останній учасник це зробить – опитування завершується.

1.3 Аналіз успішних ІТ-проєктів

1.3.1 Аналіз відомих технічних рішень

Загалом, існує багато можливих варіантів для автоматизації процесів в ІТ-компаніях. Серед них можна виділити наступні: десктопні та мобільні додатки, веб-додатки та боти на платформах месенджерів.

Десктопні та мобільні додатки надають найбільше можливостей в плані кількості автоматизованих задач, але часто мають високий поріг входження і потребують встановлення. Розширення функціоналу таких додатків досить складне і затратне по часу, та потребує завантаження оновлення з боку користувача.

Веб-додатки зазвичай набагато простіші у використанні в цілях автоматизації, ніж десктопні додатки, але використання надто великої кількості вебсайтів для автоматизації може ускладнити процес онбордингу нових працівників в компанії.

Боти (або чатботи) мають досить високу популярність серед рішень для автоматизації задач в компаніях. У порівнянні з іншими варіантами, боти зазвичай мають дуже простий інтерфейс для роботи, який можна освоїти протягом декількох хвилин.

В ІТ-компаніях одним з основних засобів комунікації є месенджери, більшість з яких мають можливості для створення та поширення чатботів. Використання ботів дозволяє поєднати процеси комунікації в компанії та

автоматизацію в одному місці – так працівникам не доведеться встановлювати додаткове ПЗ або відкривати ще одну вкладку в веб-браузері.

Наразі найпопулярнішами корпоративними месенджерами з підтримкою чатботів є Microsoft Teams та Slack. В Україні також є досить поширеним месенджер Telegram. Серед інших з можливістю створення та встановлення ботів також розповсюджені Discord та WeChat.

Telegram є одним з найпопулярніших месенджерів серед молоді (18-30 років) в Україні, що перетинається з середнім віком більшості працівників в сфері ІТ в нашій країні [3]. Саме через це Telegram часто обирають в якості основного засобу зв'язку в невеликих та середніх за розміром ІТ-компаніях.

Telegram повністю безкоштовний і має широкі можливості API для розробників ботів, що спричинило популярність цього месенджера в якості платформи для чатботів. Але більшість з цих ботів призначені для особистого користування, і корпоративних ботів які б виконували задачі пов'язані з роботою ІТ-компаній досить мало.

Discord є ще одною платформою з можливістю створення ботів, API якої дозволяє ботам навіть приймати участь в голосових дзвінках. Discord має спільну проблему з Telegram – цей сервіс дуже рідко використовується в ІТ-компаніях, оскільки він орієнтований переважно на геймерів, через що корпоративних ботів на цій платформі майже не існує.

WeChat – найпопулярніший месенджер в КНР, який використовується практично в усіх сферах життя громадян Китаю, включаючи роботу ІТ-компаній. Цікавим є факт, що можливості для створення ботів в WeChat з'явилися на декілька років раніше, ніж в інших месенджерах – в 2013 році, коли як в Telegram ця можливість з'явилася в 2015 [4].

Однак, в WeChat сильно обмежена навіть можливість реєстрації користувачів, якщо вони не з материкового Китаю або не мають знайомих з китайським мобільним номером, що унеможливлює його використання в більшості компаній. Однак, цей месенджер та боти на його платформі є

непоганим вибором для працівників ІТ-компаній, що працюють разом з китайськими командами або на замовників з КНР.

Slack – один з найперших месенджерів, що був створений саме для корпоративних цілей. Основна ціль Slack – підвищення ефективості процесів комунікації між працівниками в компаніях, що досягається завдяки потужному функціоналу для упорядкування переписок (створення різних каналів, чатів, «ниток» повідомлень) та зручному пошуку по всіх повідомленнях. Кількість активних користувачів в Slack станом на 2019 і 2020 роки сягає 12 мільйонів щоденно, а кількість організацій, що використовують цей месенджер – більше 600 000 [5].

Завдяки можливостям Slack API, у відкритому доступі налічується більше 800 різних інтеграцій та ботів на цій платформі. Slack API має обширну документацію і SDK для більшості сучасних мов програмування, а боти на основі цього API можуть не лише відповідати на повідомлення текстом і картинками, а й створювати цілі інтерактивні блоки з кнопками і формами для вводу даних.

Серед недоліків Slack варто виділити обмеженість безкоштовної версії у порівнянні з аналогами, наприклад кількості встановлених ботів чи збережених повідомлень в історії.

Microsoft Teams з’явився як відповідь Microsoft на популярність Slack. В 2016 році керівництво корпорації роздумувало над придбанням Slack за 8 мільярдів доларів, але врешті було прийнято рішення про створення власного інструменту для корпоративного месенджингу. Станом на 2019 та 2020 роки, Microsoft Teams встиг обігнати Slack за кількістю активних користувачів, і наразі налічує понад 75 мільйонів активних користувачів на добу [6].

API для розробки ботів в Microsoft Teams має схожі можливості з Slack, однак SDK для деяких популярних мов програмування не оновлювалися більше року на момент написання цього дипломного проєкту. Також, через свою новизну у порівнянні з Slack та Telegram, Microsoft Teams не зважаючи на велику кількість користувачів налічує досить мало ботів у відкритому доступі. З іншого

боку, цей месенджер має офіційну інтеграцію з більшістю сервісів Microsoft таких як Office 365, тож він є найкращим вибором якщо в IT-компанії вже використовуються такі сервіси.

Зважаючи на всі переваги та недоліки різних платформ та технологій для автоматизації задач, було обрано саме бота для Slack, оскільки цей месенджер вже використовується у багатьох IT-компаніях з 10-50 працівників, для яких дана розробка призначена в першу чергу. Через свою популярність протягом останніх 5 років, в інтернеті накопичилася велика кількість ресурсів, що можуть допомогти в розробці бота для Slack – відповіді на StackOverflow, бібліотеки для спрощення роботи з Slack API, а також велика кількість ботів у відкритому доступі, аналізуючи котрі можна покращити кінцевий продукт та пришвидшити розробку.

1.3.2 Аналіз відомих програмних продуктів

На базі платформи Slack існує порівняно велика кількість корпоративних ботів.

Standuply – корпоративний чатбот для Slack, що дозволяє автоматизувати окремі процеси менеджменту проєктів, зокрема:

- організація стендап-мітингів та ретроспектив;
- збір фідбеку;
- створення голосувань;
- проведення “планувального покеру”;
- перегляд беклогу задач.

Оскільки Standuply призначений конкретно для менеджменту проєктів та не має механізмів для розширення з боку компанії, варіанти використання даного бота дещо обмежені.

Бот розповсюджується за платною щомісячною підпискою, ціна на яку встановлюється в залежності від кількості людей в команді та списку потрібного функціоналу. Команди розміром до 4 чоловік можуть користуватися ботом

безкоштовно [7]. Серед недоліків Standuply згідно можна виділити високу вартість підписки, складність налаштування та освоєння.

Kyber – ще один корпоративний Slack-бот, що спрощує організацію роботи в команді. В функціонал цього бота входить:

– менеджмент проєктів в окремих Slack-каналах;

– планування та моніторинг задач;

– організація стендап-мітингів;

– створення голосувань та опитувань;

– створення нагадувань;

– створення та використання “мікро-програм” (без написання програмного коду, візуально) для автоматизації.

Як і Standuply, Kyber був створений для менеджменту проектів, а можливості з розширення не дозволяють додати принципово новий функціонал, як наприклад автоматизацію обліку відпусток. Через це бот може не підійти деяким компаніям.

Схема розповсюдження Kyber – платна щомісячна підписка, ціна на яку залежить від кількості людей в команді. Для особистого використання без команди бот є безкоштовним [8]. Серед недоліків Kyber – складність освоєння та висока вартість.

Vacation Tracker – корпоративний бот, що автоматизує процеси пов’язані з оформленням відпусток. Користувачі бота можуть створювати запити на вихід у відпустку чи на відгули, а менеджери та адміністратори стверджують чи відхиляють ці запити. Також цей бот може надсилати сповіщення всім користувачам, коли хтось вже вийшов у відпустку. Всі параметри (доступні типи відгулів, кількість доступних днів, вигляд та частота сповіщень) можна налаштувати в адміністративній панелі бота.

Як і попередні два боти, Vacation Tracker фокусується на конкретній задачі, а саме веденні відпусток чи лікарняних, і не має можливостей з роширення функціоналу на інші задачі.

Vacation Tracker має безкоштовний випробувальний період, після якого компанія повинна сплачувати щомісячну підписку для подальшого використання [9]. Серед недоліків – більшість параметрів бота можна налаштувати лише в веб-версії додатка, що дещо ускладнює роботу. Також до недоліків можна віднести високу вартість підписки навіть для малих команд.

Hubot – бот для Slack з відкритим програмним кодом, створений компанією GitHub Inc. для автоматизації будь-яких робочих процесів. Сам по собі Hubot містить лише базовий функціонал (інтеграція з Google Maps, переклад, пошук зображень), однак архітектура цього бота дозволяє встановлення скриптів на мові CoffeeScript, за допомогою яких можливо автоматизувати безліч інших задач. Це дозволяє компаніям персоналізувати бота, додаючи потрібний функціонал [10].

Під час аналізу цього аналога, було знайдено готові скрипти для інтеграції з Jira, що може частково автоматизувати процеси менеджменту проєктів, а також декілька скриптів для проведення опитувань та голосувань.

На жаль, велика кількість скриптів для Hubot на GitHub вже не підтримується, а нові скрипти з’являються все рідше через дуже низьку популярність як мови CoffeeScript, так і самого бота.

Порівняльну характеристику зазначених ботів наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

	Standuply	Kyber	Vacations Tracker	Hubot
Менеджмент проєктів	+	+	-	±
Проведення опитувань	+	+	-	+

Продовження таблиці 1.1

Оформлення відпусток	-	-	+	-
Можливість розширення	-	±	-	+
Схема розповсюдження	$	$	$	Open-source

1.4 Аналіз вимог до програмного забезпечення

В даній предметній області можна виділити двох наступних акторів:

– звичайний працівник компанії (далі Користувач);

– менеджер або працівник відділу HR (далі Адміністратор).

Також, з огляду на можливості для розширення бота шляхом додавання нових модулів, варто виділити актора Розробник. Слід зазначити, що один працівник компанії зможе водночас виконувати різні ролі – наприклад, адміністратор зможе виступати в ролі користувача, а користувач в ролі розробника.

Авторизація та реєстрація в боті не потребується, оскільки вже виконана в Slack. Механізм надання ролі адміністратора забезпечно шляхом надання доступу до закритого каналу в Slack, тож на стороні бота даний функціонал не потрібен.

Варіанти використання наведені у наступних таблицях:

Таблиця 1.2 – Варіант використання UC001

Назва	Створення запиту на відпустку
Опис	Будь-який користувач може створити запит на відпустку чи відгул

Продовження таблиці 1.2

Учасники	Користувач
Передумови	
Постумови	Запит збережено в історії користувача та відправлено адміністратору
Основний сценарій	Користувач вводить в чат команду для створення запиту Бот відкриває модальне вікно в Slack Користувач вводить дату початку відпустки Користувач вводить дату закінчення відпустки Користувач вводить причину Користувач натискає на кнопку підтвердження

Таблиця 1.3 – Варіант використання UC002

Назва	Схвалення запиту на відпустку
Опис	Адміністратор може схвалити запит на відпустку
Учасники	Адміністратор
Передумови	Користувач створив запит на відпустку
Постумови	Користувач отримав сповіщення що його запит схвалено Запит оновлено в історії користувача
Основний сценарій	Адміністратор натискає на кнопку для схвалення запиту

Таблиця 1.4 – Варіант використання UC003

Назва	Відхилення запиту на відпустку
Опис	Адміністратор може відхилити запит на відпустку
Учасники	Адміністратор
Передумови	Користувач створив запит на відпустку
Постумови	Користувач отримав сповіщення що його запит відхилено Запит оновлено в історії користувача
Основний сценарій	Адміністратор натискає на кнопку для відхилення запиту

Таблиця 1.5 – Варіант використання UC004

Назва	Перегляд власної статистики відпусток
Опис	Користувач може переглянути власну статистику відпусток чи відгулів
Учасники	Користувач
Передумови	
Постумови	Користувач отримав кількість доступних днів та свою історію відпусток
Основний сценарій	Користувач вводить в чат команду для виведення статистики

Таблиця 1.6 – Варіант використання UC005

Назва	Перегляд статистики відпусток
Опис	Адміністратор може переглянути статистику відпусток чи відгулів будь-якого користувача
Учасники	Адміністратор
Передумови	
Постумови	Адміністратор отримав історію відпусток обраного користувача та кількість доступних йому днів по кожному типу відгулів
Основний сценарій	Адміністратор вводить в чат команду для виведення статистики та ім’я користувача

Таблиця 1.7 – Варіант використання UC006

Назва	Додавання доступних днів
Опис	Адміністратор може додати доступні дні на будь-який тип відгулів користувачам
Учасники	Адміністратор
Передумови	
Постумови	Адміністратор додав доступні дні на обраний тип відгулу користувачу
Основний сценарій	Адміністратор вводить в чат команду для для додавання днів, кількість днів та тип відгулу

Таблиця 1.8 – Варіант використання UC007

Назва	Створення опитувальника
Опис	Користувач може створити опитувальник та відправити його іншим користувачам
Учасники	Користувач
Передумови	
Постумови	Опитувальник відправлено обраним користувачам
Основний сценарій	Користувач вводить в чат команду для створення опитувальника і кількість питань Користувач вводить назву опитувальника Користувач обирає отримувачів опитувальника Користувач вводить всі питання Користувач натискає на кнопку підтвердження

Таблиця 1.9 – Варіант використання UC008

Назва	Введення відповідей до опитування
Опис	Користувач може відповісти на питання в опитувальнику
Учасники	Користувач
Передумови	Користувачу відправили опитування
Постумови	Автор опитувальника отримав відповіді на питання

Продовження таблиці 1.9

Основний сценарій	Користувач натискає на кнопку відповіді Користувач вводить відповіді на запропоновані запитання Користувач натискає на кнопку підтвердження

Таблиця 1.10 – Варіант використання UC009

Назва	Додавання нових модулів
Опис	Розробник може додати новий модуль до бота
Учасники	Розробник
Передумови	
Постумови	Функціонал модуля доступно користувачам чи адміністраторам
Основний сценарій	Розробник вказує назву модуля, опис, описує команди та інтерактивні події, реалізує функціонал Модуль додається до бота за допомогою API

Таблиця 1.11 – Варіант використання UC010

Назва	Взаємодія модулів зі Slack API
Опис	Розробник може використовувати можливості Slack API в модулях
Учасники	Розробник
Передумови	Модуль додано до бота

Продовження таблиці 1.11

Постумови	Модуль може відправляти повідомлення, інтерактивні елементи та виконувати інші дії з Slack API
Основний сценарій	Розробник вказує назву метода Slack API та параметри Модуль виконує запит за допомогою API

Таблиця 1.12 – Варіант використання UC011

Назва	Взаємодія користувачів з модулями
Опис	Користувач може виконувати команди і працювати з інтерактивними компонентами з різних модулей бота
Учасники	Користувач
Передумови	Розробник додав модуль до бота
Постумови	Користувач зміг виконати взаємодію з одним з модулів бота
Основний сценарій	Користувач вводить в чат команду або взаємодіє з інтерактивним компонентом (кнопка, форма для введення даних тощо) Бот викликає потрібний метод в обраному модулі за допомогою API

Таблиця 1.13 – Варіант використання UC012

Назва	Отримування сповіщень про користувачів у відпустці

Продовження таблиці 1.13

Опис	Користувач може отримувати сповіщення коли інші користувачі вийшли у відпустку чи відгул
Учасники	Користувач
Передумови	Користувач зайшов у канал Slack зі сповіщеннями
Постумови	Користувач отримує сповіщення кожного дня коли хтось виходить у відпустку
Основний сценарій	Бот надсилає повідомлення в спеціальний канал Slack кожного дня зі списком людей у відпустці

Таблиця 1.14 – Варіант використання UC013

Назва	Повторна відправка опитувальника
Опис	Користувач може повторно відіслати опитування учасникам, які ще не заповнили відповіді
Учасники	Користувач
Передумови	Користувач створив опитувальник, є учасники які не пройшли цей опитувальник
Постумови	Учасники, які ще не заповнили відповіді, отримують повторне нагадування

Продовження таблиці 1.14

Основний сценарій	Користувач натискає на кнопку для повторної відправки опитування. Бот надсилає сповіщення всім учасникам, які ще не пройшли опитування

1.4.1 Розроблення функціональних вимог

Функціональні вимоги до бота описані наступними таблицями:

Таблиця 1.15 – Опис функціональної вимоги REQ001

Номер	REQ001
Назва	Створення запитів на отримання відпусток чи відгулів
Опис	Користувач має змогу створити запит на отримання відпустки чи відгулу в обрану дату. Бот повинен зберегти цей запит в історії та направити його адміністраторам

Таблиця 1.16 – Опис функціональної вимоги REQ002

Номер	REQ002
Назва	Перегляд власної історії відпусток та відгулів
Опис	Користувач має змогу переглянути власну історію відпусток та відгулів

Таблиця 1.17 – Опис функціональної вимоги REQ003

Номер	REQ003
Назва	Перегляд доступних днів кожного типу відгулів

Продовження таблиці 1.17

Опис	Користувач має змогу переглянути кількість доступних йому днів по кожному типу відгулів. Дні можуть додаватися адміністраторами або нараховуватися з певною періодичністю щоденно.

Таблиця 1.18 – Опис функціональної вимоги REQ004

Номер	REQ004
Назва	Отримання сповіщень якщо хтось з користувачів у відпустці
Опис	Користувач має змогу отримувати сповіщення коли інші працівники компанії виходять у відпустку чи відгул. Бот повинен надсилати сповіщення кожного дня в спеціальний канал Slack, якщо хоч один користувач в цей день у відпустці.

Таблиця 1.19 – Опис функціональної вимоги REQ005

Номер	REQ005
Назва	Створення опитувальників та розсилання іншим користувачам
Опис	Користувач має змогу створити опитувальник з своїми запитаннями та відправити його на заповнення іншим користувачам. Бот має зберегти цей опитувальник та відправити сповіщення користувачам, що мають його пройти

Таблиця 1.20 – Опис функціональної вимоги REQ006

Номер	REQ006
Назва	Нагадування учасникам опитування що треба заповнити відповіді

Продовження таблиці 1.20

Опис	Користувач може повторно надіслати нагадування, якщо учасники надто довго не відповідають. Бот при цьому заново відправляє запитання тим учасникам, які ще не заповнили відповіді

Таблиця 1.21 – Опис функціональної вимоги REQ007

Номер	REQ007
Назва	Заповнення відповідей в опитувальниках
Опис	Користувач може заповнити відповіді на запитання, якщо йому було надіслано опитування

Таблиця 1.22 – Опис функціональної вимоги REQ008

Номер	REQ008
Назва	Схвалення та відхилення запитів на отримання відпусток чи відгулів
Опис	Адміністратор може схвалити або відхилити запит на отримання відпустки. При цьому бот зберігає результат в історії користувача і відправляє повідомлення з результатом користувачу

Таблиця 1.23 – Опис функціональної вимоги REQ009

Номер	REQ009
Назва	Перегляд історії відпусток чи відгулів всіх користувачів
Опис	Адміністратор може переглянути історію відпусток чи відгулів будь-якого користувача.

Таблиця 1.24 – Опис функціональної вимоги REQ010

Номер	REQ010

Продовження таблиці 1.24

Назва	Додавання користувачам доступних днів по кожному типу відгулів
Опис	Адміністратор може додавати будь-якому користувачу додаткові доступні дні по кожному з типів відгулів.

Таблиця 1.25 – Опис функціональної вимоги REQ011

Номер	REQ011
Назва	Можливість розширення функціоналу
Опис	Розробник може розширити функціонал бота під потреби компанії шляхом додавання нових модулів, які можуть взаємодіяти зі Slack API. Користувач має змогу використовувати функціонал цих модулів за допомогою команд або взаємодії з інтерактивними компонентами.

Матрицю залежності варіантів використання та функціональних вимог наведено на рисунку 1.3.

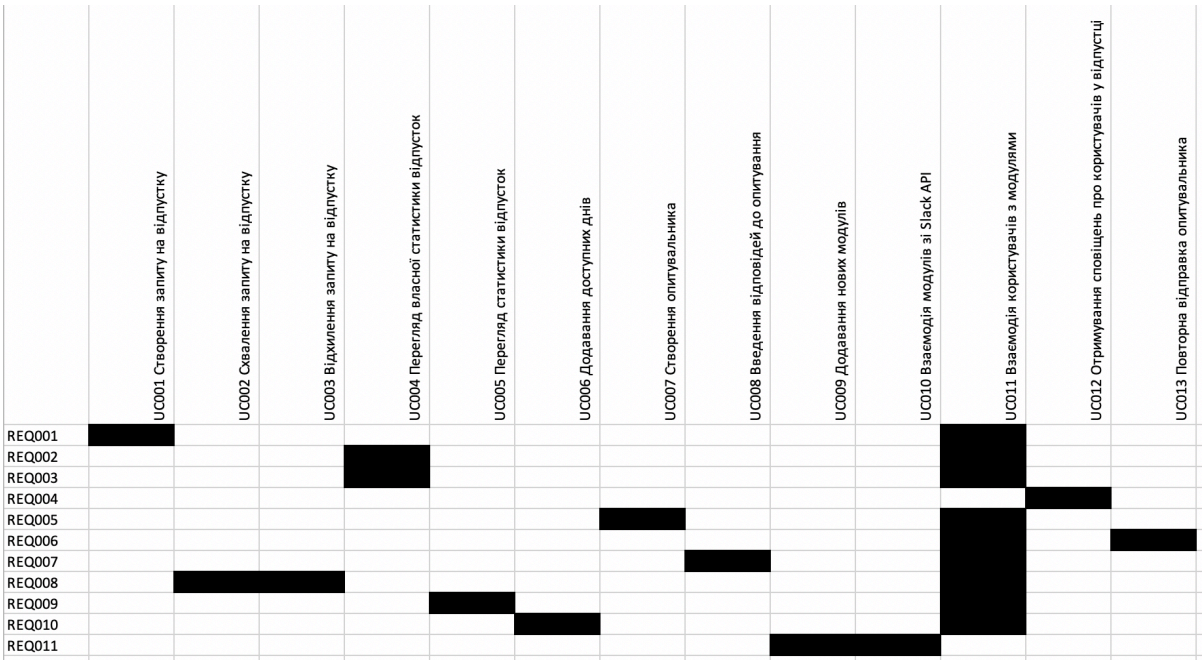


Рисунок 1.3 – Матриця трасування

1.4.2 Розроблення нефункціональних вимог

Створений бот має задовільняти наступні нефункціональні вимоги:

– формат даних для передачі інформації між модулями бота, головним сервером і Slack – JSON;

– взаємодія між головним сервером бота і модулями відбувається по протоколу HTTP з використанням архітектури REST;

– мова повідомлень бота – англійська;

– при отримуванні запитів з боку Slack потрібно перевіряти цілісність інформації за допомогою секретного ключа та механізму HMAC;

– обмін даних між Slack та ботом повинен використовувати TLS (HTTPS);

– для створення модулів бота потрібно реалізувати бібліотеку, яка спрощує опис команд, інтерактивних компонентів, додавання модуля та взаємодію з Slack;

– бот повинен розповсюджуватися за ліцензією MIT а його код має бути доступним на Github або інших сервісах.

1.4.3 Постановка комплексу завдань модулю

Розроблене програмне забезпечення створюється для підвищення ефективності роботи та зменшення витрат IT-компаній малого та середнього розміру.

Мета створення даної роботи – автоматизація рутинних процесів в IT-компаніях за допомогою модульного бота в Slack, що має декілька вбудованих модулів (модуль для оформлення відпусток та модуль проведення опитувань) та можливість додавати нові модулі для автоматизації інших процесів компанії.

Для досягнення поставленої мети, застосунок повинен вирішувати наступні задачі:

– інтеграція головного сервера бота та Slack API;

– API для додавання нових модулів;

– інтерфейс для користувачів в Slack для взаємодії зі всіма модулями бота;

– довідка для користувачів з інформацією про всі модулі бота;

– модуль відпусток: обробка запитів на оформлення відпусток, надання статистики користувачам та адміністраторам (кількість доступних днів, історія проведених відпусток), сповіщення користувачам коли хтось вийшов у відпустку;

– модуль опитувань: формування опитувальників з заданими питаннями, розсилка сповіщень про необхідність пройти опитування, збір відповідей учасників опитування.

Сервер та модулі бота повинні працювати на серверах під управлінням ОС Linux, а інтерфейс користувача доступний на всіх платформах, які підтримує Slack (Windows, Linux, MacOS, Android, iOS, Windows Phone, веб-додаток).

1.5 Висновки по розділу

Автоматизація рутинних процесів в IT-компаніях є актуальним питанням, оскільки це дозволяє значно підвищити ефективність роботи в компанії. В результаті порівняння підходів та платформ для автоматизації, для розробки даного дипломного проєкту було обрано бота для корпоративного месенджера Slack, оскільки саме такий формат взаємодії для автоматизації задач є найбільш зручним у порівнянні з десктоп- та веб-додатками, а його можливості для створення ботів дозволяють реалізувати максимально зручний інтерфейс для працівників компаній.

Було проведено аналіз існуючих корпоративних ботів для Slack і виявлено, що більшість з них не підходять під потреби малих за розміром компаній, оскільки використання більш ніж одного такого бота сильно збільшить щомісячні витрати компанії. Також, використання запропонованих ботів може бути складним в плані адаптації в компаніях, оскільки функціонал може частково

не відповідати потребам, а можливості з розширення переважно відсутні або значно обмежені через закритість коду.

З огляду на це, розробка корпоративного бота Slack з можливістю динамічного розширення функціоналу для розробників, відкритим програмним кодом та простим інтерфейсом для працівників була б дуже корисною для IT-компаній. Можливість створення нових модулів та завантаження існуючих з Github дозволить адаптувати бота під потреби великої кількості компаній, зокрема малі та середні за розміром.

В рамках даної дипломної роботи потрібно розробити наступні компоненти:

– головний сервер бота, що виконує взаємодію з Slack API та направляє команди користувачів на потрібний модуль;

– модуль оформлення відпусток та відгулів, що дозволяє автоматизувати процес створення та схвалення запитів працівників;

– модуль проведення опитувань, що дозволяє працівникам автоматизувати процес створення опитувальників та збір відповідей.

Додатково буде реалізовано невелику бібліотеку-помічник для створення нових модулів бота, а також вбудований модуль для виведення довідки по іншим модулям.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Перед початком проєктування архітектури та власне створення цього бота, варто проаналізувати взаємодію користувача з ботом та змоделювати бізнес-процеси використання. Як було зазначено в попередньому розділі, ПЗ буде складатся з трьох основних компонентів: головний сервер, що взаємодіє з користувачем за допомогою Slack API; модуль оформлення відпусток; модуль проведення опитувань.

Для початку роботи з кожним модулем, користувачу потрібно буде викликати команду (відправити повідомлення з символом «/» на початку), або провзаємодіяти з певним інтерактивним компонентом (наприклад, кнопкою в одному з попередніх повідомлень бота).

Детальну структурну схему бізнес-процесів використання бота зображено за специфікацією BPMN в графічній частині.

2.2 Архітектура програмного забезпечення

2.2.1 Загальна архітектура бота та алгоритм роботи

При аналізі можливих варіантів архітектури додатку, було обрано паттерн мікросервісної архітектури у спрощеному вигляді. Таке рішення є доцільним, оскільки кожен модуль бота фактично не залежить від інших, через що може розглядатись як окремий сервіс. Додатково, вибір мікросервісної архітектури набагато спростить розширення бота, адже цей паттерн надає наступні можливості:

– можливість оновлювати окремі модулі бота, не зачіпаючи інші;

– різні розробники чи команди можуть працювати над різними модулями одночасно;

– можливість використовувати різні мови програмування, фреймворки, бібліотеки та бази даних для кожного модуля;

– помилка в одному модулі бота не вплине на роботу інших.

Звичайно ж, є і недоліки використання мікросервісів в даному випадку у порівнянні з монолітною архітектурою – дещо ускладнюється процес розгортання нових модулів на сервері, однак це можна легко вирішити за допомогою автоматизації.

Для спілкування між різними сервісами обрано протокол HTTP та REST API з передачою даних у форматі JSON. Це є одним з найпоширеніших варіантів через свою простоту та знайомість протоколу більшості розробників. Повну схему взаємодії мікросервісів, баз даних та сторонніх API зображено на рисунку 2.1. В якості чата-платформи для бота в першому розділі було обрано Slack, взаємодія з яким відбувається на головному сервері. Інші модулі (мікросервіси) можуть отримати доступ до існуючих методів Slack API шляхом взаємодії з головним сервером.

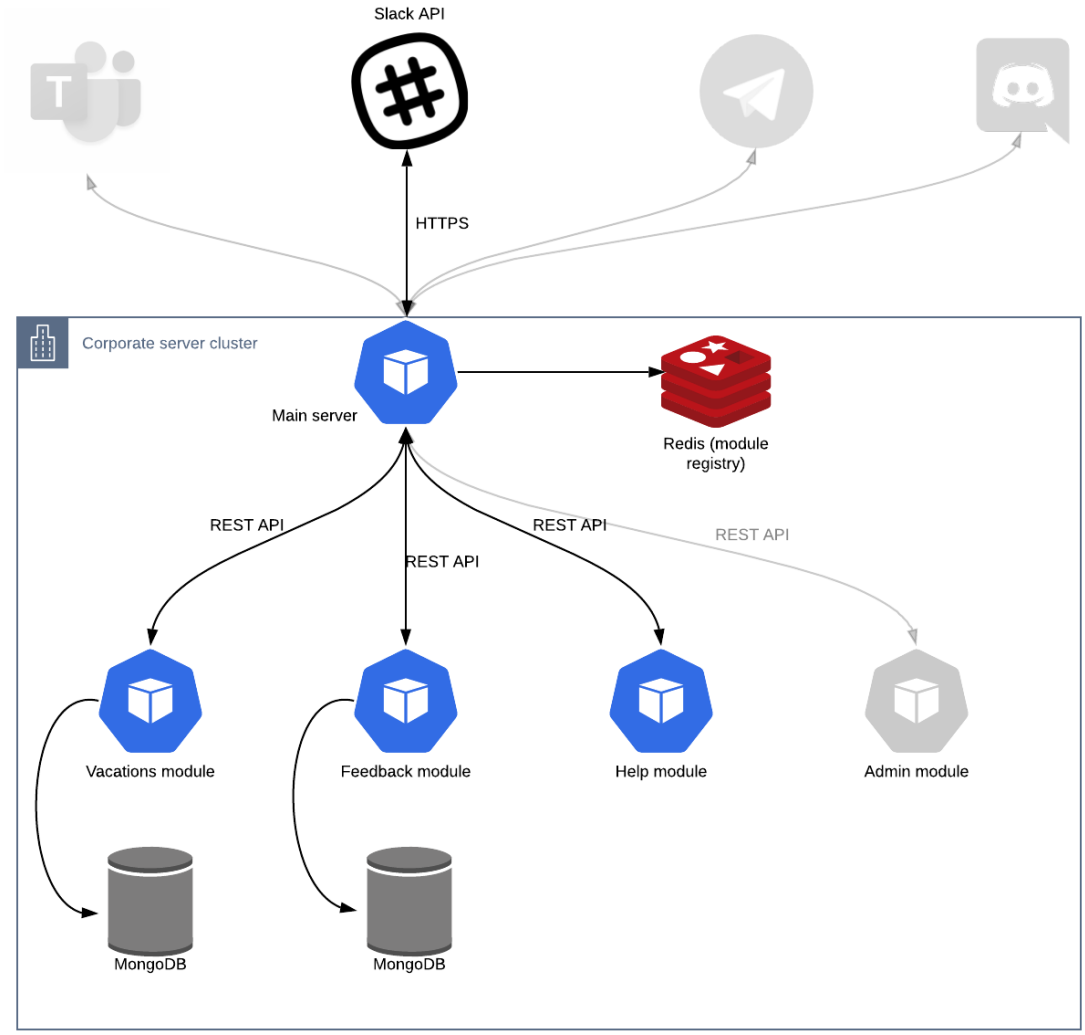


Рисунок 2.1 – Загальна схема архітектури бота

Методи API головного серверу бота наведено в таблиці 2.1.

Таблиця 2.1 – Методи API головного серверу бота

Метод HTTP	Назва	Опис метода
GET	/api/modules/	Метод для отримання повної інформації про всі модулі, зареєстровані на сервері
GET	/api/modules/{name}	Метод для отримання інформації по конкретному модулю за його назвою

Продовження таблиці 2.1

POST	/api/modules/	Метод додавання інформації про новий модуль
POST	/api/slack/	Метод для виклику Slack API з боку модулів бота
POST	/slack/commands/	Метод для вхідних команд з боку Slack API. Викликається коли користувач ввів команду в чат
POST	/slack/actions/	Метод для вхідних інтерактивних подій з боку Slack API. Викликається коли користувач натиснув на кнопку в одному з повідомлень бота чи заповнив форму

Задля більшої гнучкості в розробці, було вирішено реєструвати нові модулі шляхом запиту модуля з інформацією про себе. Це дозволяє додавати нові модулі автоматично та без редагування коду чи конфігураційних файлів головного сервера. Так, кожен модуль повинен повідомляти інформацію про себе з інтервалом в 10-20 секунд за допомогою методу POST на шлях /api/modules/ головного сервера (зазвичай такий процес називається heartbeat [11]). Якщо модуль певний час не відсилав heartbeat (наприклад, коли сталася програмна чи апаратна помилка), головний сервер автоматично видаляє його з реєстру та перстає надавати користувачам можливість взаємодії з ним. Модель JSON для передачі інформації про модуль детальніше описана у підрозділі 2.2.2.

Для коректної роботи бота, кожен модуль повинен надавати стандартизовані API, що наведені в таблиці 2.2.

Таблиця 2.2 – Методи API модулів бота

Метод HTTP	Назва	Опис метода
POST	/api/commands/{name}	Метод для виконання команд з даного модуля, викликається головним сервером коли від користувача надходить коректна команда
POST	/api/actions/{action_id}	Метод для виконання подій, викликається коли користувач натиснув на кнопку чи інший інтерактивний компонент, на який підписаний цей модуль

В загальному, алгоритми обробки команд та інтерактивних подій на головному сервері можна описати наступним чином.

а) **КОЛИ** отримано команду від користувача:

1) розділити команду на назву модуля, назву команди та аргументи;

2) **ЯКЩО** назву модуля не надано або модуля з такою назвою не існує, повідомити користувача про помилку;

3) **ЯКЩО** назву команди не надано або команди з такою назвою в модулі не існує, повідомити користувача про помилку;

4) **ЯКЩО** не надано потрібної кількості обов’язкових аргументів команди, повідомити користувача про помилку;

5) викликати потрібну команду на сервері модуля з заданими аргументами;

6) **ЯКЩО** виникла помилка під час з’єднання з модулем, повідомити користувача.

б) **КОЛИ** отримано інтерактивну подію від користувача:

1) з’ясувати тип події (компонент повідомлення чи відправка форми);

2) знайти модуль, що підписаний на цю подію;

3) **ЯКЩО** існує такий модуль, викликати подію на сервері модуля.

Дані алгоритми реалізовано в методах класів CommandDispatcher та ActionDispatcher, відповідно. Детальніше ці методи описано в підрозділі 2.2.4.

Для спрощення реалізації модулів, вирішено створити бібліотеку-помічник на обраній мові програмування, яка додає необхідні шляхи API на сервері, дозволяє декларативно описувати команди чи події і виконує механізм heartbeat до головного сервера. За допомогою методів класу Module створеної бібліотеки, розробники можуть додавати функції для обробки команд чи подій, а детальний список доступних команд та подій передається на головний сервер разом з запитом heartbeat. Детальніше методи цього класу описано в підрозділі 2.2.4, а керівництво користування бібліотекою наведене в «Керівництво програміста».

За допомогою створеної бібліотеки, було реалізовано потрібні сервіси, алгоритми роботи яких навдено нижче.

Модуль відпусток. Даний сервіс відповідає за обробку команд та подій, що пов’язані з оформленням відпусток. Алгоритм роботи модулю наведений нижче.

а) **КОЛИ** отримано команду створення запиту на відпустку:

1) **ЯКЩО** користувач не ввів дату початку відпустки, відкрити форму для заповнення даних про відпустку;

2) **ЯКЩО** формат запису дати початку чи кінця не відповідає ISO, повідомити користувача про помилку;

3) виконати створення запиту на відпустку (пункт в).

б) **КОЛИ** отримано результати форми про відпустку:

1) виконати створення запиту на відпустку (пункт в).

в) Створення запиту на відпустку:

1) **ЯКЩО** не надано тип відпустки або надано невірний тип відпустки, вивести помилку;

2) **ЯКЩО** не надано дату кінця відпустки, встановити дату початку відпустки (відгул на 1 день);

3) **ЯКЩО** дата початку пізніша ніж дата кінця, обміняти значення змінних;

4) зберегти запит в базу даних;

5) відправити повідомлення користувачу, що його запит створено;

6) відправити повідомлення адміністраторам з інформацією про запит та кнопками схвалення і відхилення.

г) **КОЛИ** натиснуто на кнопку схвалення запиту:

1) оновити статус запиту в базі даних;

2) зменшити кількість доступних днів у користувача;

3) відправити повідомлення користувачу що його запит схвалено.

д) **КОЛИ** натиснуто на кнопку відхилення запиту:

1) оновити статус запиту в базі даних;

2) відправити повідомлення користувачу що його запит відхилено.

е) **КОЛИ** отримано команду на перегляд статистики:

1) **ЯКЩО** користувач не ввів ім’я користувача, використати його власне ім’я;

2) **ЯКЩО** введено ім’я іншого користувача, а в поточного немає прав адміністратора, вивести помилку;

3) знайти всі записи в базі відпусток створені цим користувачем;

4) знайти в базі запис з кількістю доступних днів в цього користувача;

5) відправити повідомлення з статистикою користувачу.

Модуль опитувань. Даний сервіс відповідає за обробку команд та подій, що пов’язані зі створенням опитувальників та збиранням на них відповіді. Алгоритм роботи модулю наведено нижче.

а) **КОЛИ** отримано команду на створення нового опитувальника:

1) **ЯКЩО** введена некоректна кількість запитань, повідомити про помилку;

2) відкрити форму для заповнення запитань для опитувальника.

б) **КОЛИ** отримано результат форми на створення нового опитувальника:

1) зберегти опитувальник в базі даних;

2) відправити повідомлення контролю стану опитування з кнопкою для повторної відправки отримувачам;

3) відправити повідомлення з кнопкою для відповіді на опитувальник всім отримувачам.

в) **КОЛИ** натиснуто на кнопку повторної відправки опитувальника:

1) знайти опитувальник в базі даних;

2) відправити повідомлення з кнопкою відповіді всім отримувачам, хто ще не відповів.

г) **КОЛИ** натиснуто на кнопку відповіді:

1) відкрити форму для заповнення відповідей на кожне з запитань в опитувальнику.

д) **КОЛИ** отримано результат форми на заповнення відповідей:

1) зберегти результати в базі даних;

2) відправити повідомлення автору опитувальника з результатами;

3) **ЯКЩО** це останній хто з отримувачів заповнив відповіді, відправити автору сповіщення про завершення.

2.2.2 Вибір та структура баз даних

Для збереження реєстру модулів на головному сервері було вирішено використати key-value NoSQL базу даних. Це надає наступні переваги порівняно зі зберіганням реєстру в пам’яті процесу сервера:

– збереження реєстру модулів після перезавантаження головного сервера (наприклад, під час оновлення);

– можливість автоматично видаляти збережені дані через певний час (для реалізації механізму автоматичної очистки неактивних модулів);

– можливість масштабувати кількість запущених процесів головного сервера під час високого навантаження.

Серед найпопулярніших key-value баз даних – Redis, Memcached та DynamoDB.

DynamoDB – пропрієтарне NoSQL сховище від Amazon, що досить часто застосовується в IoT. Зазвичай цю БД обирають коли вже користуються іншими веб-сервісами від Amazon. DynamoDB дуже зручна у використанні, майже не потребує налаштування (оскільки повністю керується платформою Amazon Web Services), може автоматично масштабуватися, однак має закритий програмний код та може дорого коштувати для компаній, що її використовують.

Memcached – розподілене сховище, основне призначення якого – кешування невеликої за обсягом інформації для пришвидшення роботи веб-додатків. Memcached проста у використанні, не програє у швидкості іншим key-value БД, має драйвери для більшості мов програмування, однак менш ефективно використовує оперативну пам’ять у порівнянні з аналогами.

Redis – мабуть, найвідоміша NoSQL база даних, що може використовуватися для кешування, зберігання даних, або в якості брокера

повідомлень. У порівнянні з іншими аналогами, Redis має набагато більшу кількість доступного функціоналу, через що має високу популярність у різних сферах [12].

Розглянуті key-value сховища майже не поступаються одне одному з точки зору продуктивності та зручності використання, тож було зроблено вибір в бік Redis через найбільшу популярність.

Було використано ключі з наступними назвами:

– **module:*** – для збереження повної інформації про модуль у форматі JSON, видаляються автоматично якщо модуль неактивний;

– **action:*** – для збереження назви модуля що підписався на даний тип подій, видаляються автоматично якщо модуль неактивний.

Структуру полів JSON, що зберігається по ключу модуля, наведено у таблиці 2.3. В цій же самій формі інформація про модуль передається в API під час процесу heartbeat, описаного в попередньому підрозділі.

Таблиця 2.3 – Структура полів JSON з інформацією про модуль

Назва поля	Тип	Опис
name	string	Назва модуля
description	string	Опис модуля (опціональний)
url	string	HTTP посилання на модуль
commands	object	JSON-об’єкт з інформацією про доступні команди (ключ – назва команди, значення – опис команди у вигляді JSON-об’єкту)
actions	array<string>	Список ідентифікаторів подій, на які підписаний цей модуль

Структуру полів JSON, що описує команди модулів, наведено у таблиці 2.4.

Таблиця 2.4 – Структура полів JSON з інформацією про команду

Назва поля	Тип	Опис
name	string	Назва команди
description	string	Опис команди (опціональний)
arguments	array<object>	Список JSON-об’єктів, що описують аргументи команди

Структуру полів JSON, що описує аргументи команд, наведено у таблиці 2.5.

Таблиця 2.5 – Структура полів JSON з інформацією про аргумент

Назва поля	Тип	Опис
name	string	Назва команди
description	string	Опис команди (опціональний)
is_optional	boolean	Булеве значення, що свідчить про опціональність даного аргументу при виконанні команди

Щодо модулів відпусток та опитувань, в якості бази даних для обох було обрано документо-орієнтовну NoSQL базу **MongoDB**. Mongo дозволяє зберігати дані у вигляді колекцій JSON-подібних документів формату BSON, а також на відміну від реляційних баз даних не має фіксованої схеми, тобто в одній колекції можуть бути присутні документи з різною структурою.

Для реалізації функціоналу зазначених модулів немає потреби у реляційних зв’язках. Також, звертаючи увагу на те, що потреби ІТ-компаній часто змінюються (наприклад, для деяких компаній потрібно буде додати нове

поле в базу), відсутність фіксованої схеми є великою перевагою на користь MongoDB, оскільки при оновленні коду та додаванні нової інформації не потрібно проводити міграції БД.

Серед документо-орієнтовних аналогів MongoDB можна виділити **CouchDB** та **DynamoDB**, які майже не поступаються в продуктивності та можливостям, однак мають значно нижчу популярність та відомість, через що для зручності було обрано саме MongoDB [13].

Структуру колекцій модуля відпусток зображено на рисунку 2.2.

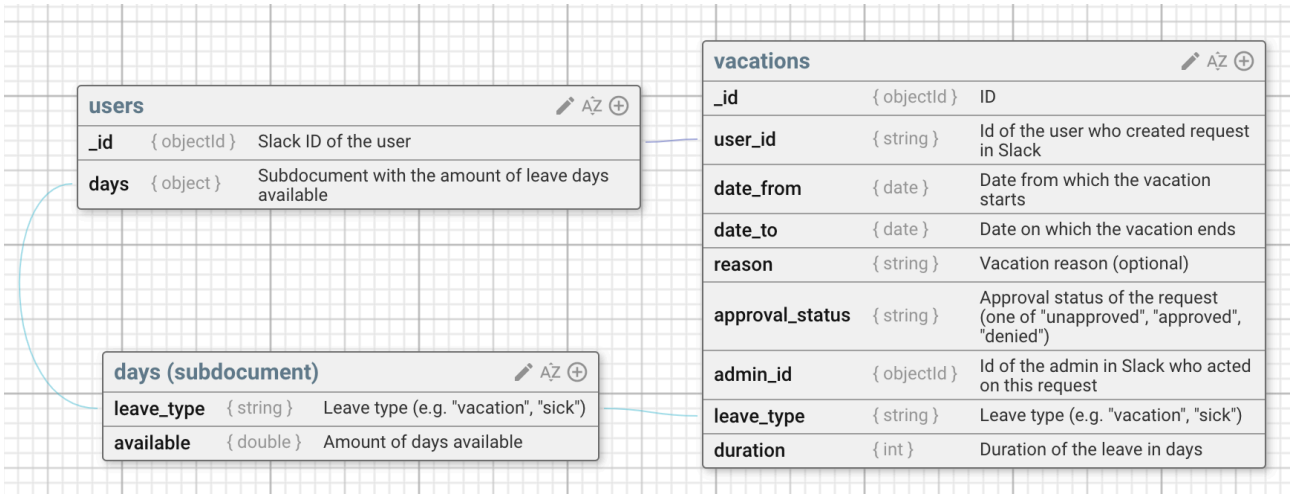


Рисунок 2.2 – Структура бази даних для модулю відпусток

В кожному документі колекції відпусток зберігаються:

– ідентифікатор запиту;
– ідентифікатор користувача Slack, що створив запит;
– дата початку відпустки;
– дата кінця відпустки;
– тривалість відпустки в днях;
– тип (відпустка, лікарняний, відгул тощо);
– причина відпустки чи відгулу;
– статус запиту;
– ідентифікатор користувача слак, що схвалив чи відмовив запит.

В кожному документі колекції користувачів зберігаються:

– ідентифікатор користувача Slack;

– піддокумент з кількістю доступних днів по кожному типу відпустки.

Цих даних достатньо для коректної роботи всіх варіантів використання модулю відпусток.

Структуру колекції для збереження опитувальників зображено на рисунку 2.3.

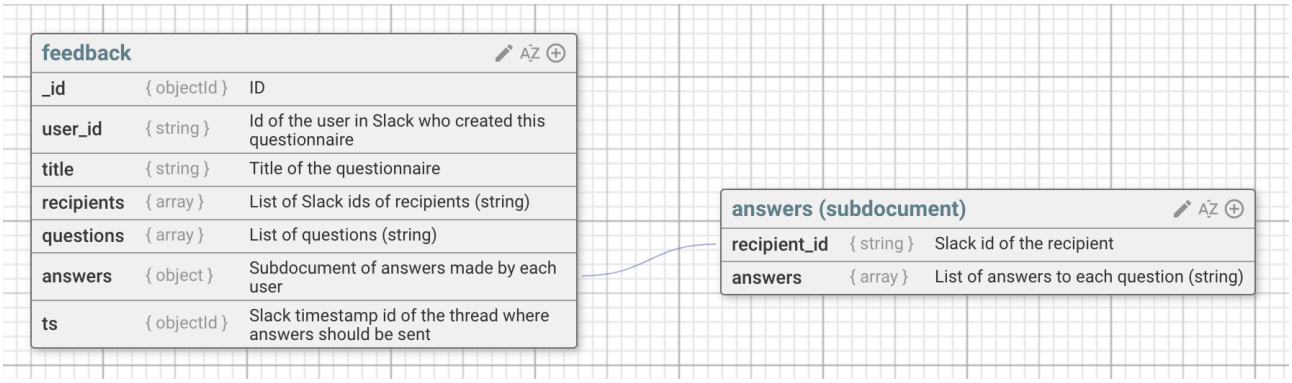


Рисунок 2.3 – Структура бази даних для модулю опитувань

В кожному документі даної колекції зберігаються:

– ідентифікатор опитувальника;

– ідентифікатор користувача Slack, що створив опитувальник;

– назву опитувальника;

– список ідентифікаторів отримувачів опитувальника;

– список запитань в опитувальнику;

– піддокумент з відповідями кожного з отримувачів;

– ідентифікатор повідомлення контролю опитування, у відповідях до якого бот буде збирати звіт.

Цих даних достатньо для коректної роботи всіх варіантів використання модулю опитувань.

2.2.3 Вибір мови програмування та бібліотек

Для реалізації головного сервера і модулів бота було обрано мову програмування **Python 3.8**. Ця мова широко використовується при створенні ботів через свою простоту, швидкість написання коду на ній та зручність при розробці. Оскільки програми на цій мові не потребують компіляції, існують додатки які дозволяють автоматично оновлювати запущену програму при будь-якій зміні коду на диску, що значно пришвидшує процес розробки та тестування.

Саме через це Python має широку підтримку в спільноті розробників ПЗ з використанням Slack API, через що можна значно спростити процес взаємодії з Slack. Так, наприклад, офіційна бібліотека **slackclient** в числі найперших з усіх мов програмування отримує оновлення при змінах в Slack API. Slackclient поширюється за ліцензією MIT, а програмний код доступний на Github, де він використовується в більше ніж 6000 проєктах. Slackclient використовує останні можливості мови Python такі як асинхронність, що забезпечує оптимальне використання ресурсів CPU під час взаємодії з мережею.

Для реалізації внутрішніх API та зворотньої взаємодії зі Slack було обрано фреймворк **FastAPI.** Це порівняно новий фреймворк для серверної розробки, що використовує інтерфейс ASGI для спілкування між сервером та кодом на Python. ASGI дозволяє серверу обробляти запити користувачів асинхронно в одному процесі, на відміну від WSGI, що використовується в фреймворках як Django і Flask і виконує всі запити синхронно [14]. Принцип роботи ASGI у спрощеному вигляді зображено на рисунку 2.4.

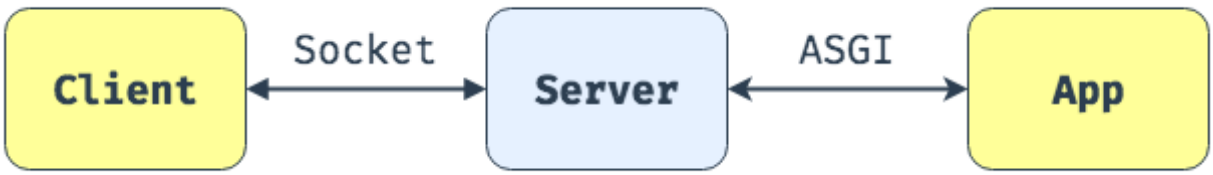


Рисунок 2.4 – Схема роботи ASGI

Асинхронність виконання дозволяє додаткам з використанням FastAPI отримувати показники швидкості, що не поступаються мовам Go та Node.js. В якості ASGI-сервера для обробки запитів було обрано **uvicorn**, за рекомендацією автора фреймворку FastAPI, оскількі цей сервер має найбільші показники швидкості у порівнянні з аналогами, а також дуже простий у налаштуванні. Uvicorn має опцію автоматичного перезавантаження сервера після оновлення файлів з кодом на диску, що значно пришвидшує розробку.

FastAPI також використовує можливості бібліотеки **Pydantic**, що дозволяє швидко описувати класи-моделі даних за допомогою анотацій типів Python. Моделі pydantic допомагають провалідувати вхідні та вихідні дані сервера, а також використовуються для автоматичної серіалізації/десеріалізації даних у форматі JSON. Завдяки цьому значно скорочується кількість повторюваного коду на валідацію вхідних даних.

За допомогою pydantic, фреймворк FastAPI автоматично генерує документацію API за специфікацією OpenAPI (Swagger), що може бути використана для відображення інтерактивної довідки для роботи з сервером або навіть генерування коду клієнтів цього API [15].

Окрім цього, для спрощення взаємодії з Slack API на головному сервері, а саме для часткового опрацювання вхідних запитів з командами та подіями, використано допоміжну бібліотеку **slackers**. Вона являє собою плагін для FastAPI, який додає методи в API сервера для доступу з боку Slack, а також виконує перевірку HMAC-підпису.

Можливості FastAPI та pydantic були використані також для написання модулів бота. В рамках дипломного проєкту було створено бібліотеку-помічник **fastapi_metabot**, за допомогою якої і реалізовані окремі модулі-сервіси бота. Бібліотеку було завантажено в репозиторій Pypi у відкритому доступі, а програмний код завантажено на Github [16]. Механізми та принцип роботи цієї бібліотеки було описано в підрозділі 2.2.1.

В якості драйвера до сховища Redis на головному сервері було використано бібліотеку **aioredis**, що надає асинхронний інтерфейс доступу до цієї бази даних. Це дозволяє далі обробляти запити користувачів та модулів в той час коли відбувається очікування серверу Redis.

За аналогічним принципом працює асинхронна бібліотека-драйвер для MongoDB – **motor**. Цю бібліотеку було використано в модулях відпусток та опитувань.

2.2.4 Опис структури програмного коду

В ході виконання дипломного проєкту було використано принципи як об'єктно-орієнтованого програмування, так і функціонального. Більшість логіки головного сервера бота та бібліотеки для створення плагінів реалізовано з використанням ООП, а плагіни бота використовують ФП з поділом на окремі модулі за призначенням функцій.

Програми на мові Python, написані у функціональному стилі, зазвичай використовують всередині функцій нефункціональні можливості (I/O операції, присвоєння змінних тощо), однак мають функціональний інтерфейс ззовні. Так, наприклад, реалізація функції може містити присвоєння локальних змінних, але не модифікує глобальний стан програми [17]. Ці принципи було використано при написанні функціональної частини ПЗ.

Схему імпортування модулів головного сервера зображено на рисунку 2.5.

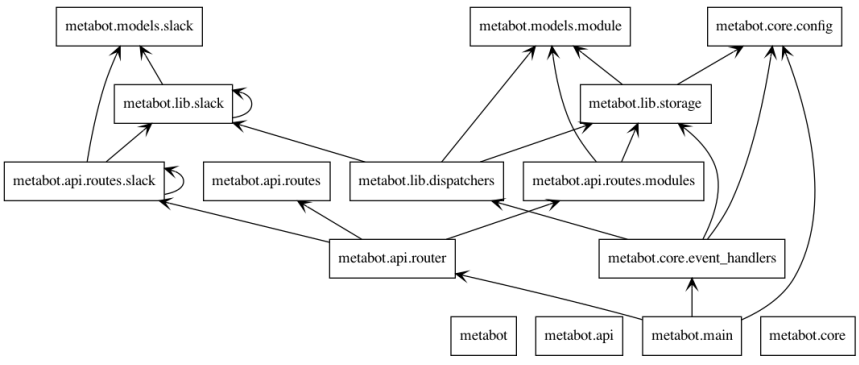


Рисунок 2.5 – Модулі головного сервера бота

Модулі поділено наступним чином:

– **metabot.api.*** – код, що відповідає за опис методів REST API;

– **metabot.core.*** – код, що відповідає за налаштування сервера (зчитування конфігурації, підключення до Redis, ініціалізація клієнту Slack API та інших об’єктів);

– **metabot.models.*** – опис моделей даних з використанням pydantic;

– **metabot.lib.*** – код, що відповідає за основну логіку роботи серверу.

Варто окремо виділити **metabot.lib.dispatchers**, в якому описано класи CommandDispatcher та ActionDispatcher, які відповідають за найголовнішу частину роботи сервера – обробка вхідних запитів з боку Slack та виклик команд і подій в модулях (мікросервісах) бота. Схему структурну цих класів та інших, з якими вони взаємодіють, зображено на рисунку 2.6.

Рисунок 2.6 – Схема структурна класів головного сервера бота

Схему імпортування модулів допоміжної бібліотеки fastapi_metabot зображено на рисунку 2.7.

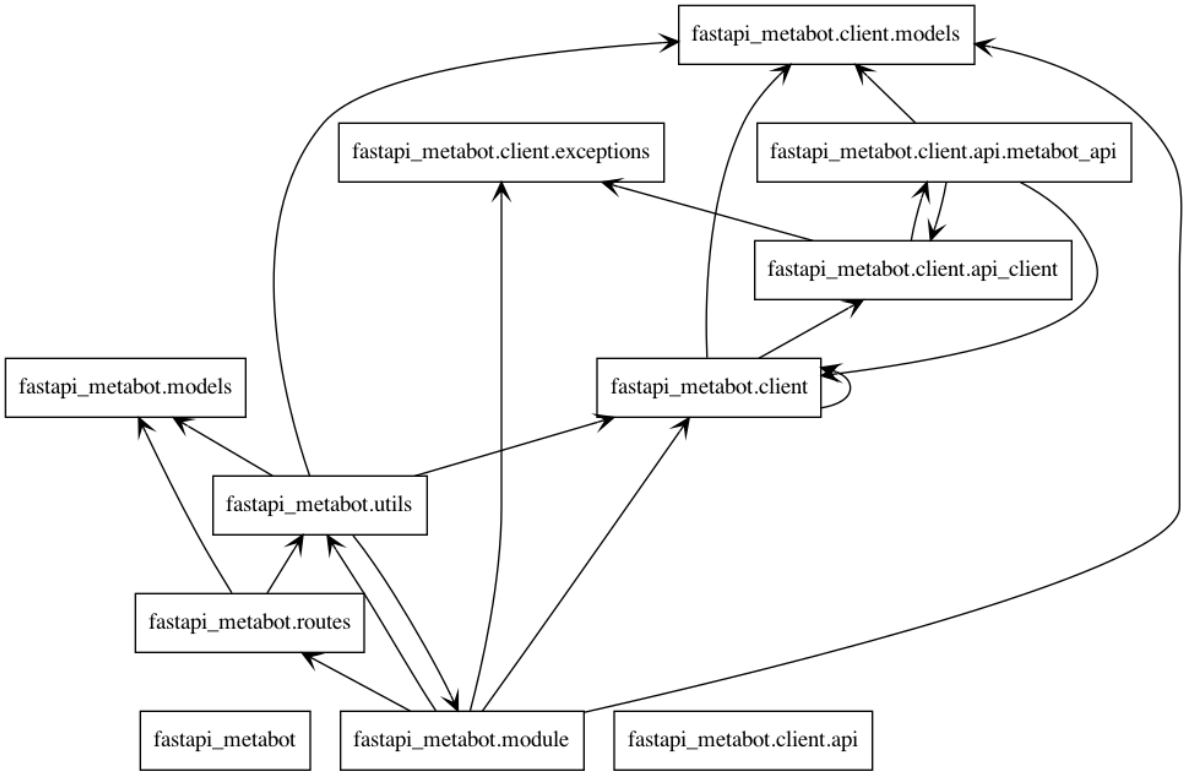


Рисунок 2.7 – Модулі бібліотеки для створення плагінів бота

Модулі в бібліотеці поділено наступним чином:

– **fastapi_metabot.module** – містить клас для створення та підключення нового модуля бота;

– **fastapi_metabot.client.*** – містить клієнтські класи для взаємодії з головним сервером (згенеровано автоматично на основі OpenAPI специфікації головного сервера);

– **fastapi_metabot.utils** – містить утиліти для розробників, такі як отримання контекстних даних про поточну команду чи подію, або відправка запиту в Slack API;

– **fastapi_metabot.routes** – містить опис методів API модуля бота (для виклику команд та подій);

– **fastapi_metabot.models** – опис моделей даних з використанням pydantic.

Основні класи бібліотеки зображено на рисунку 2.8.

Рисунок 2.8 – Схема структурна класів бібліотеки для створення плагінів

Схему імпортування модулів в сервісах відпусток та опитувань зображено на рисунку 2.9.

Рисунок 2.9 – Модулі сервісів відпусток та опитувань

Модулі в сервісах поділено наступним чином:

– ***.db** – код для взаємодії з базою даних

– ***.bl** – бізнес-логіка сервісу

– ***.main** – опис доступних команд, подій

– ***.config**, ***.event_handlers** – налаштування сервера

– ***.builders** – допоміжні функції для побудови інтерактивних повідомлень і форм Slack

Детальний опис основних функцій і методів з вхідними та вихідними даними наведено у таблиці 2.6.

Таблиця 2.6 – Основні функції і методи програми

№	Назва	Опис	Вхідні дані	Вихідні дані
1.	metabot.lib.dispatchers.CommandDispatcher.dispatch	Переадресовує команди користувачів Slack на відповідні модулі	Інформація про команду з Slack	Відсутні
2.	metabot.lib.dispatchers.ActionDispatcher.dispatch	Переадресовує інтерактивні події користувачів Slack на відповідні модулі	Інформація про подію з Slack	Відсутні
3.	metabot.lib.slack.slack_request	Виконує запит до Slack API	Клієнт Slack API, назва методу, дані запиту	Відповідь Slack API
4.	metabot.lib.storage.Storage.add_or_replace_module	Зберігає або перезаписує інформацію про модуль бота в Redis з терміном деактивації	Інформація про модуль	Відсутні

Продовження таблиці 2.6

5.	metabot.lib.storage.Storage.get_module	Отримання модуля бота за назвою	Назва модуля	Інформація про модуль
6.	metabot.lib.storage.Storage.get_all_modules	Отримання списку всіх активних модулів бота	Відсутні	Інформація про всі модулі у вигляді списку
7.	metabot.lib.storage.Storage.get_module_by_action	Отримання модуля за подією	Ідентифікатор події	Інформація про модуль який підписаний на цю подію
8.	fastapi_metabot.utils.slack_request	Виконання запиту до Slack API для модулів бота	Назва методу, дані запиту	Відповідь Slack API
9.	fastapi_metabot.utils.async_slack_request	Виконання асинхронного запиту до Slack API для модулів бота	Назва методу, дані запиту	Відповідь Slack API
10.	fastapi_metabot.utils.get_current_user_id	Отримання ідентифікатора поточного користувача Slack, що взаємодіє з модулем бота	Відсутні	Ідентифікатор користувача

Продовження таблиці 2.6

11.	fastapi_metabot.utils.get_current_channel_id	Отримання ідентифікатора поточного каналу Slack, в якому відбувається взаємодія з ботом	Відсутні	Ідентифікатор каналу
12.	fastapi_metabot.module.Module.command	Метод для додавання нової команди до модуля, може бути використаний як декоратор	Назва, опис, опис аргументів, функція для обробки команди	Відсутні
12.	fastapi_metabot.module.Module.action	Метод для додавання нової події до модуля, може бути використаний як декоратор	Ідентифікатор, функція для обробки події	Відсутні
12.	fastapi_metabot.module.Module.view	Метод для додавання нової форми до модуля, може бути використаний як декоратор	Ідентифікатор, функція для обробки форми	Відсутні
13.	fastapi_metabot.module.Module.execute_command	Метод для виконання команди в модулі бота	Назва команди, аргументи	Відсутні
14.	fastapi_metabot.module.Module.execute_action	Метод для виконання події чи форми в модулі бота	Ідентифікатор події	Відсутні

Продовження таблиці 2.6

15.	fastapi_metabot.module.Module.install	Встановлення модуля бота на сервер	Об’єкт серверу FastAPI	Відсутні
16.	fastapi_metabot.module.Module._start_heartbeat	Початок heartbeat модуля, викликається на старті сервера	Відсутні	Відсутні
17.	fastapi_metabot.module.Module._stop_heartbeat	Припинення heartbeat модуля, викликається на завершенні роботи сервера	Відсутні	Відсутні
18.	feedback.bl.open_creation_view	Відкриття форми для створення опитувальника	Кількість запитань	Відсутні
19.	feedback.bl.create_questionnaire	Створення опитувальника	Назва, список запитань, список отримувачів	Відсутні
20.	feedback.bl.submit_answers	Відправка відповідей до опитувальника	Ідентифікатор опитувальника, відповіді	Відсутні
21.	feedback.bl.notify_recipients	Відправка сповіщення отримувачам опитувальника	Ідентифікатор опитувальника	Відсутні

Продовження таблиці 2.6

22.	feedback.bl.open_answer_view	Відкриття форми для заповнення відповідей опитувальника	Ідентифікатор опитувальника	Відсутні
23.	vacations.bl.open_request_view	Відкриття форми для створення запиту на відпустку	Відсутні	Відсутні
24.	vacations.bl.create_request	Створення запиту на відпустку	Дата початку, дата кінця, причина	Відсутні
25.	vacations.bl.is_admin_channel	Перевірка прав адміністратора у користувача	Відсутні	True якщо є права адміністратора
26.	vacations.bl.process_request	Оновлення статусу запиту на відпустку	Ідентифікатор запиту, новий статус	Відсутні
27.	vacations.bl.send_history	Відправка історії відпусток користувачу	Ідентифікатор користувача	Відсутні

2.3 Конструювання програмного забезпечення

Для мови Python наразі існує безліч допоміжних засобів, що можуть спростити та пришвидшити процес написання коду. Декілька таких засобів було використано в рамках конструювання даного дипломного проєкту.

PyCharm Professional. PyCharm – одне з найбільш популярних IDE для мови Python, створене розробниками IntelliJ IDEA – компанією JetBrains. PyCharm значно полегшує навігацію в коді навіть великих проєктів, а також проводить повну індексацію коду проєкту «на льоту» з метою надання підсказок, автодоповнення, виявлення помилок та проблем зі стилем коду. Професійна версія PyCharm доступна студентам ВНЗ безкоштовно.

Flake8. Flake8 – утиліта, яка перевіряє програмний код проєкту на відповідність стандартам та стилю коду (linter). Flake8 поєднує в собі декілька перевірок, таких як відповідність коду до офіційного стандарту PEP8, пошук помилок в коді, проблем з відступами, перевірка на складність функцій та методів. Ця утиліта досить проста в установці та налаштуванні, дозволяє покращити якість коду і зекономити час на розробку.

Муру. Оскільки Python – мова з динамічною типізацією, розробники часто допускають помилки при використанні змінних в коді (наприклад, виклик методу над об’єктом зі значенням None призведе до помилки AttributeError, що є досить частою помилкою). В 2014 році з’явився стандарт PEP 484, що ввів синтакс для анотації змінних та функцій типами. Разом з цим з’явилися утиліти для статичної перевірки на типи, однією з яких є Муру, що використовують ці анотації для перевірки коректності використання змінних чи функцій. Це дозволяє відловити помилки в програмі ще до її запуску, якщо розробник анотував всі функції в коді.

Також в ході розробки було використано засоби, які не є специфічними лише для Python. Серед таких – **Docker** та **docker-compose**.

Docker дозволяє розробникам створювати та запускати так звані контейнери, що містять програму або сервер разом з усіма бібліотеками і налаштуваннями. Використання контейнерів дає можливість розробляти і запускати код на всіх операційних системах однаково, без виникнення проблем з встановленням залежностей. Було створено образи контейнерів, що містять

головний сервер бота, та окремі образи для кожного модуля, а також використано стандартні контейнери з базами даних Redis та MongoDB.

Docker-compose – програмне забезпечення, що дозволяє поєднати декілька окремих Docker-контейнерів в одному файлі та запустити їх разом. Зважаючи на те, що даний дипломний проєкт використовує паттерн мікросервісної архітектури, для повноцінної роботи проєкту потрібно запустити близько 6 контейнерів, тож docker-compose значно спрощує цей процес і дозволяє швидко та ефективно працювати з проєктом в ході локальної розробки.

В якості системи контролю версій для коду програми було використано **Git** у поєднанні з **Github**.

2.4 Аналіз безпеки даних

Для коректної роботи з Slack API, будь-який бот має відкрити доступ «в світ» до декількох методів API. Якщо цього не зробити, бот не зможе отримувати вхідні запити з боку Slack, коли користувач взаємодіє з ботом. Однак, якщо не потурбуватися про безпеку, це відкриває декілька векторів атак. Наприклад, зловмисники зможуть підробити запит Slack і викликати будь-яку команду бота, з можливістю зміни користувача, який це зробив.

В документації Slack зазначено два способи захистити канал зв’язку. **Перший** – перевірка токена, який надсилається разом з кожним запитом, шляхом звичайного порівняння (якщо значення співпадає зі значенням в налаштуванням в конфігурації сервера, то запит вважається справжнім). Однак, цей метод вже вважається застарілим та ненадійним, оскільки зловмисники можуть перехопити запит та використати цей токен в своїх цілях, тож офіційна документація Slack не рекомендує такий спосіб.

Другий спосіб – перевірка підпису, що надсилається з кожним запитом, за допомогою секретного ключа. Створення та перевірка підпису відбувається за механізмом HMAC, що використовує криптографічну хеш-функцію (наприклад, SHA256) для створення «підпису» на основі повідомлення та секретного ключа.

За допомогою цього підпису можна перевірити не тільки автентичність інформації, а і її цілісність (тобто, зловмисник не зможе змінити дані в повідомленні). Детальний процес роботи HMAC зображено на рисунку 2.10 [18]. Саме цей спосіб було використано під час розробки даної дипломної роботи, адже його рекомендовано в офіційній документації Slack [19].

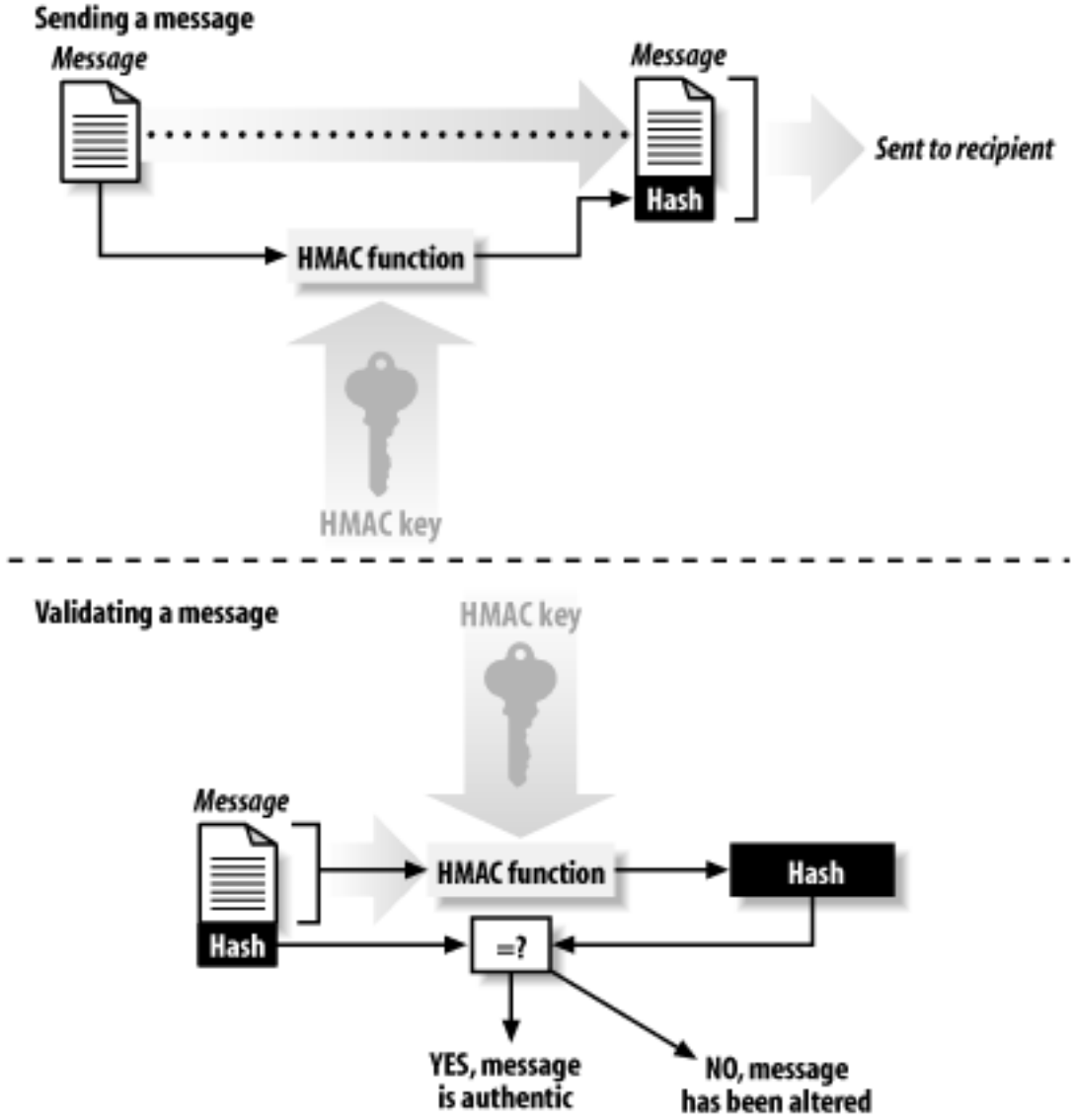


Рисунок 2.10 – Принцип роботи HMAC

Також, для покращення безпеки даного програмного забезпечення, було використано TLS (HTTPS) для зв’язку між Slack та головним сервером. Це унеможливлює перехоплення трафіку зловмисниками, що підвищує безпеку та

конфіденційність даних IT-компанії, яка використовує бота. Для створення і підписання криптографічного сертифіката, який потрібен для роботи TLS, використано центр сертифікації Let’s Encrypt.

Для підвищення рівня безпеки збережених даних, всі взаємодії з базами даних MongoDB та Redis захищені логіном та паролем. Ці логіни та паролі, а також інші секретні значення (наприклад, ключ доступу бота до Slack API, ключ для перевірки підпису) винесено в змінні оточення, що робить їх недоступними в системі версіонування та в Docker-образах.

2.5 Висновки по розділу

В даному розділі було проаналізовано бізнес-процеси корпоративного бота, детально спроєктовано архітектуру з урахуванням можливості розширення функціоналу та інших вимог. Було обрано мову програмування, фреймворки, бібліотеки та бази даних, що потрібні для реалізації проєкту. Було проаналізовано бота з точки зору безпеки даних та додано механізми захисту інформації від зловмисників.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вступ та призначення плану тестування

Для аналізу якості та відповідності програмного забезпечення вимогам, було створено план тестування за стандартом IEEE 829 [20]. Цей план тестування ставить за мету наступне.

– Визначити інструменти для проведення тестування.

– Визначити вимоги до оточення, в якому потрібно проводити тестування.

– Визначити як саме мають проходити тести.

3.2 Предмети тестування

Тестуванню підлягають вся серверна частина бота (головний сервер та модулі), бібліотека для створення модулів і інтерфейс користувача бота в Slack.

3.3 Функціонал, що підлягає тестуванню

Тестуванню підлягають наступні функції корпоративного бота:

– як розробник, додавання нових модулів;

– як користувач, створення запиту на відпустку чи відгул;

– як користувач, перегляд власної статистики відпусток та відгулів;

– як користувач, створення опитувальника;

– як користувач, можливість відповіді на опитувальник;

– як адміністратор, схвалення запитів на відпустку;

– як адміністратор, відхилення запитів на відпустку.

3.4 Функціонал, що не підлягає тестуванню

Оскільки бот реалізовано на платформі Slack, яка розробляється компанією Slack Technologies, Inc., функціонал цієї платформи (наприклад, можливість

відправки повідомлень або введення команд) не входить в рамки даного плану тестування. Однак, якщо проблеми чи недоліки платформи не дозволяють використовувати основний функціонал бота, слід звернутися в офіційну підтримку Slack.

3.5 Підхід до тестування

3.5.1 Статичне тестування на якість коду

Статичне тестування на якість коду (lint) має проводитись після будь-якої зміни в коді, та перевіряти відповідність коду стандарту PEP8, стиль коду та правильність використання типів. Для статичного тестування потрібно використовувати інструменти **flake8** та **mypy**, конфігураційні файли яких повинні знаходитися в корні репозиторія на Github.

3.5.2 Модульне тестування

Модульне (unit) тестування повинно виконуватися після будь-якої зміни в коді, а код модульних тестів повинен знаходитися в репозиторії Github разом з кодом бота чи його модулів. Для написання модульних тестів варто використовувати бібліотеку **pytest.** Модульне тестування повинно перевіряти наступне:

- робота окремих методів API головного сервера;
- робота окремих функцій бібліотеки для створення модулів;
- робота методів API, що створюються бібліотекою;
- робота окремих функцій кожного модуля бота.

При розробці нового функціоналу варто використовувати принцип TDD (test-driven development) та покривати тестами принаймні 80% рядків коду.

3.5.3 Інтеграційне тестування

Інтеграційне тестування повинно виконуватися перед оновленням версії бота на сервері та перевіряти наступне:

– взаємодію модулів бота та головного сервера;
– взаємодію модулів бота та бази даних;
– взаємодію головного сервера та Slack API.

3.5.4 Функціональне тестування

Тестування функціоналу бота повинно проводитися згідно до варіантів використання, наведених у розділі 1. В разі знайдення дефекту, потрібно створити Issue на Github проєкту з детальним описом проблеми.

3.5.5 Тестування на продуктивність

Тестування на продуктивність повинно відбуватись одночасно з функціональним тестуванням шляхом порівняння часу між виконанням дії та отриманням відповіді від бота.

3.6 Критерії проходження тестування

3.6.1 Тестування на якість коду

Тестування на якість коду вважається успішним тільки якщо не інструменти flake8 та mypy не знайшли жодної проблеми.

3.6.2 Модульне тестування

Модульне тестування вважається успішним тільки за умови 100% проходження всіх тестів та якщо code coverage >= 80%.

3.6.3 Інтеграційне тестування

Інтеграційне тестування вважається успішним тільки за умови 100% проходження всіх тестів.

3.6.4 Функціональне тестування

Функціональне тестування вважається успішним за умови коректної роботи всіх варіантів використання. Допускаються невеликі розбіжності між результатами тестування та описом варіантів використання, якщо вони не сильно впливають на можливість користування ботом.

3.6.5 Тестування на продуктивність

Тестування на продуктивність вважається успішним за умови часу відповіді бота менше 3 секунд. У разі перевищення цього порогу, перестануть працювати деякі компоненти бота через особливості роботи Slack API.

3.7 Критерії припинення тестування

Тестування повинно бути зупинене в разі виникнення помилок під час тестування якості коду, модульного чи інтеграційного тестування. Тестування варто припинити, якщо є проблеми з роботою Slack API або програми Slack.

3.8 Вимоги до середовища

Серверна частина повинна тестуватися на машині під управлінням Linux (бажано Debian-подібній, наприклад Ubuntu) з технічними характеристиками, зазначеними в технічному завданні або кращими.

Користувацький інтерфейс бота повинен бути протестований в офіційних додатках Slack на будь якій мобільній платформі (Android, iOS) та на одній з десктопних платформ (Windows, macOS, Linux). Для тестування потрібно мати аккаунт Slack та додати його в команду, де вже встановлено бота.

3.9 Опис тестів

Опис функціональних тест-кейсів до корпоративного бота наведено у таблиці 3.1. Всі тест-кейси було успішно пройдено, а результати співпадають з очікуваними, що свідчить про якість програмного забезпечення.

Таблиця 3.1 – Опис функціональних тестів

Назва	Передумови	Кроки відтворення	Очікуваний результат	Результат
Створення запиту на відпустку (неправильний тип відпустки)	В конфігурації немає типу відпустки “asd”	Ввести команду: “/meta vacations request asd”	Отримано помилку зі списком можливих типів	Отримано помилку зі списком можливих типів
Створення запиту на відпустку (команда)	В конфігурації є тип відпусток “sick”	Ввести команду: “/meta vacations request sick TODAY TODAY reason” (де TODAY – сьогоднішня дата)	Запит створено	Запит створено
Створення запиту на відпустку (форма)	Немає	Ввести команду: “/meta vacations request”. Заповнити всі поля форми	Запит створено	Запит створено
Схвалення запиту на відпустку	Створено запит, запит ще не оброблено	Відкрити канал адміністраторів, натиснути на кнопку “Approve” поряд з запитом	Запит схвалено	Запит схвалено
Схвалення обробленого запиту	Створено запит, запит вже оброблено	Відкрити канал адміністраторів, натиснути на кнопку “Approve”	Отримано помилку	Отримано помилку

Продовження таблиці 3.1

Відхилення запиту на відпустку	Створено запит, запит ще не оброблено	Відкрити канал адміністраторів, натиснути на кнопку “Deny” поряд з запитом	Запит відхилено	Запит відхилено
Відображення статистики відпусток	Немає	Ввести команду: “/meta vacations stats”	Отримано статистику	Отримано статистику
Створення опитувальника	Немає	Ввести команду: “/meta feedback create 3”, заповнити форму	Опитувальник відправлено отримувачам	Опитувальник відправлено отримувачам
Відповідь на опитувальник	Створено опитувальник	Натиснути на кнопку “Answer”, заповнити форму	Відповіді відправлено автору	Відповіді відправлено автору
Введення неіснуючого модулю	Не існує модуля з назвою “test”	Ввести команду: “/meta test asd”	Отримано помилку	Отримано помилку
Введення неіснуючої команди	Немає	Ввести команду: “/meta help test”	Отримано помилку	Отримано помилку

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Оскільки дане програмне забезпечення є ботом для Slack, користувачам для розгортання потрібно лише встановити на свій пристрій офіційний додаток Slack та отримати запрошення до команди, в якій вже встановлено та налаштовано бота.

Для розгортання серверної частини бота можна використати один з наступних методів:

– ручне розгортання (на «чистому залізі» або в хмарі, без контейнеризації);

– docker-compose (для розгортання на одному сервері чи локальній машині);

– Docker Swarm (для розгортання на кластері серверів);

– Kubernetes (для розгортання на кластері серверів).

При локальній розробці бота та нових модулів рекомендується використовувати **docker-compose** для зручності (файл для запуску сервера та всіх модулів “docker-compose.yaml” можна знайти в Github репозиторії). Для запуску серверу цим методом потрібно ввести в термінал команду “**docker-compose up --build metabot help vacations feedback**”: це запустить головний сервер бота, 3 модулі, Redis та MongoDB. Для підключення з боку Slack, сервер має мати зовнішню IP-адресу або домен, який потрібно вказати в налаштуваннях Slack (при локальній розробці можна скористатися ssh-тунелем на хмарний сервер, **ngrok** або **telepresence**).

Для розгортання бота в production краще застосувати **Kubernetes** або **Docker Swarm**. У випадку з Docker Swarm, для запуску потрібно налаштувати кластер згідно до офіційної документації та використати файл “docker-compose.yaml” з Github-репозиторію бота. Якщо зроблено вибір на користь

Kubernetes, потрібно налаштувати кластер та запустити команду “**kubectl apply -f k8s/**”.

Ручне розготання не рекомендується через складність архітектури – для повноцінної роботи бота потрібно налаштувати, встановити залежності та запустити 4 сервери і 2 СКБД.

Детальну схему структурну розгортання серверної частини бота зображено в графічній частині.

4.2 Робота з програмним забезпеченням

Керівництво користувача описане у «Керівництво користувача». Керівництво створення нових модулів описане у «Керівництво програміста».

ВИСНОВКИ

У ході дипломного проєкту було проаналізовано предметну область автоматизації процесів в IT-компаніях, розглянуто існуючі рішення та продукти-аналоги.

Було спроєктовано і розроблено модульний корпоративний Slack-бот, що складається з 4 частин-мікросервісів: головний сервер, модуль оформлення відпусток, модуль проведення опитувань, модуль довідки по роботі з ботом. Для спрощення розширення функціоналу в майбутньому, розроблено допоміжну бібліотеку для написання нових модулів.

Також було описано тест-план за стандартом IEEE 829, за яким проведено аналіз якості і тестування даного програмного забезпечення.

Наведено інструкції з розгортання серверної частини бота, а також детальну схему розгортання. Інструкції користувача та програмування нових модулів описано в додатках.

Розроблений корпоративний бот вже готовий до використання в багатьох малих та середніх IT-компаніях, а завдяки розробленим механізмам розширення функціоналу його можна підлаштувати під потреби будь-якої компанії.

ПЕРЕЛІК ПОСИЛАНЬ

1) Software development in Ukraine: 2019-2020 IT market report [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.n-ix.com/software-development-in-ukraine-2019-2020-market-report/.

2) Workplace Automation is Everywhere [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.businessnewsdaily.com/9835-automation-tech-workforce.html.

3) Исследование украинского Telegram [Електронний ресурс]. – 2019 – Режим доступу до ресурсу: https://vctr.media/ukrainskiy-telegram-5000-15716/.

4) China, WeChat, and the Origins of Chatbots [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://chatbotsmagazine.com/china-wechat-and-the-origins-of-chatbots-89c481f15a44.

5) Slack Statistics [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://expandedramblings.com/index.php/slack-statistics/.

6) Microsoft Teams now has 75 million daily active users [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.businessinsider.com/microsoft-teams-hits-75-million-daily-active-users-2020-4.

7) Project Management Digital Assistant [Електронний ресурс] – Режим доступу до ресурсу: https://standuply.com/.

8) Task management, Project Management and Calendar for Slack | Kyber [Електронний ресурс] – Режим доступу до ресурсу: https://kyber.me/.

9) Vacation Tracker | Slack App Directory [Електронний ресурс] – Режим доступу до ресурсу: https://slack.com/apps/A1L3ZBM7G-vacation-tracker.

10) Hubot is your friendly robot sidekick. [Електронний ресурс] – Режим доступу до ресурсу: https://hubot.github.com/.

11) Heartbeat (computing) [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Heartbeat_(computing).

12) Amazon DynamoDB vs. Memcached vs. Redis [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://db-engines.com/en/system/Amazon+DynamoDB%3BMemcached%3BRedis.

13) Amazon DynamoDB vs. CouchDB vs. MongoDB [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://db-engines.com/en/system/Amazon+DynamoDB%3BCouchDB%3BMongoDB.

14) Introduction to ASGI: Emergence of an Async Python Web Ecosystem [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://florimond.dev/blog/articles/2019/08/introduction-to-asgi-async-python-web/.

15) FastAPI [Електронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/.

16) vasyukvv42/metabot: Modular Slack bot [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/vasyukvv42/metabot.

17) Functional Programming HOWTO [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.python.org/3/howto/functional.html.

18) HMAC. // Practical UNIX and Internet Security, 3rd Edition / , 2011. – С. 187–194.

19) Verifying requests from Slack [Електронний ресурс] – Режим доступу до ресурсу: https://api.slack.com/authentication/verifying-requests-from-slack.

20) How to Write a Test Plan with the IEEE 829 Standard [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://reqtest.com/testing-blog/how-to-write-a-test-plan-2/.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

КОРПОРАТИВНИЙ БОТ ДЛЯ ІТ-КОМПАНІЙ

Технічне завдання

КПІ.ІП-6305.045490.02.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ А.А. Коротенко

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.В. Васюк

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Корпоративний бот для ІТ-компаній

Галузь застосування: Малі та середні за розміром компанії у сфері ІТ

Наведене технічне завдання поширюється на розробку корпоративного бота для ІТ-компаній, котра використовується для автоматизації рутинних процесів та призначена для малих та середніх за розміром компаній у сфері ІТ. Користувачами можуть бути будь-які працівники та розробники ІТ-компаній.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки корпоративного бота для ІТ-компаній є завдання на дипломне проєктування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім.Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для компаній в сфері IT, та інших компаній що використовують в якості засобу для комунікації Slack.

Метою розробки є автоматизація рутинних процесів в IT-компаніях за допомогою взаємодії з ботом в Slack, а саме оформлення відпусток чи відгулів і проведення опитувань серед співробітників, з можливістю для розробників додавати нові модулі для автоматизації інших процесів компанії.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

- створення запитів на отримання відпусток чи відгулів;
- перегляд власної історії відпусток та відгулів;
- перегляд доступних днів кожного типу відгулів;
- отримання сповіщень якщо хтось з користувачів у відпустці;
- створення опитувальників та розсилання іншим користувачам;
- нагадування учасникам опитувальника що треба заповнити відповіді;
- заповнення відповідей в опитувальниках;

4.1.1.2 Для адміністратора системи:

- схвалення та відхилення запитів на отримання відпусток чи відгулів;
- перегляд історії відпусток та відгулів всіх користувачів;
- додавання користувачам доступних днів по кожному типу відгулів;

4.1.1.3 Для розробника:

- можливість розширення функціоналу для автоматизації.

4.1.2 Розробку виконати на платформі Linux

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються.

4.3.2 Обслуговування та обслуговуючий персонал.

Не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на серверах з операційною системою Ubuntu.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору

Процесори Intel i3 або кращі.

4.4.2.2 Об‘єм ОЗП

512 Мб, або більше.

4.5 Вимоги до інформаційної та програмної сумісності

– Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Unix;

– Вхідні дані повинні бути представлені в наступному форматі: тексти повідомлень Slack;

– Результати повинні бути представлені в наступному форматі: повідомлення в Slack та колекції MongoDB у форматі BSON;

– Програмне забезпечення повинно надавати REST API для підключення нових модулів.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати установочну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

а) Супроводжувальна документація:

1) пояснювальна записка;

2) керівництво користувача;

3) керівництво програміста;

4) програма та методика тестування.

б) Довідникова документація:

1) всі розроблені API повинні бути задокументовані за специфікацією OpenAPI;

2) програмне забезпечення повинно мати вбудовану довідку.

в) Графічна документація:

1) схема структурна бізнес-процесів.

2) схема структурна розгортання.

3) креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	15.03.2020	
2.	Розробка технічного завдання	25.03.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	29.03.2020	Специфікації програмного забезпечення
4.	Проєктування структури програмного забезпечення, проєктування компонентів	20.04.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	30.04.2020	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05.2020	Тести, результати тестування
7.	Розробка матеріалів текстової частини проєкту	24.05.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проєкту	17.05.2020	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	24.05.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

КОРПОРАТИВНИЙ БОТ ДЛЯ ІТ-КОМПАНІЙ

Опис програми

КПІ.ІП-6305.045490.03.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ А.А. Коротенко

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.В. Васюк

Київ – 2020 року

Тексти програмного коду

Корпоративний бот для IT-компаній

(Найменування програми (документа))

DVD-R

(Вид носія даних)

19 арк, 66 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2020

```
APP_TITLE = 'MetaBot'
API_PREFIX = '/api'
SLACKERS_PREFIX = '/slack'

config = Config('.env')

SLACK_SIGNING_SECRET = config('SLACK_SIGNING_SECRET')
SLACK_API_TOKEN = config('SLACK_API_TOKEN')
REDIS_URL = config('REDIS_URL')
MODULE_EXPIRATION_SECONDS = config(
  'MODULE_EXPIRATION_SECONDS',
  cast=int,
  default=30,
)

log = logging.getLogger(__name__)

def start_app_handler(app: FastAPI) -> Callable:
  async def startup() -> None:
    app.state.session = ClientSession()
    app.state.slack = WebClient(
      token=SLACK_API_TOKEN,
      run_async=True,
      session=app.state.session,
    )
    app.state.redis = await aioredis.create_redis_pool(REDIS_URL)
    app.state.storage = Storage(app.state.redis)

    app.state.command_dispatcher = CommandDispatcher(app)
    commands.on('meta', app.state.command_dispatcher.dispatch)

    app.state.action_dispatcher = ActionDispatcher(app)
    for action in (
      'block_actions',
      'message_actions',
      'view_submission',
      'view_closed',
    ):
      actions.on(action, app.state.action_dispatcher.dispatch)

  return startup

def stop_app_handler(app: FastAPI) -> Callable:
  async def shutdown() -> None:
    app.state.redis.close()
    await app.state.redis.wait_closed()

    await app.state.session.close()
    # Wait 250 ms for the underlying SSL connections to close
    await asyncio.sleep(0.250)

  return shutdown

SAMPLE_MODULE = {
  'name': 'help',
  'description': 'Help module',
  'url': 'http://help-module:8000',
  'commands': {
    'me': {
      'name': 'me',
      'description': 'Get help',
      'arguments': [
        {
          'name': 'module_name',
          'is_optional': True,
          'description': 'Module name'
```

```
            }
          ]
        }
    },
    'actions': [],
}

class CommandArgument(BaseModel):
    name: str = Field(regex=r'[\w\d]+', min_length=1, max_length=255)
    is_optional: bool = False
    description: Optional[str]

class Command(BaseModel):
    name: str = Field(regex=r'[\w\d]*', max_length=255)
    description: Optional[str]
    arguments: List[CommandArgument] = Field(default_factory=list)

    @validator('arguments')
    def no_required_args_after_optional(
            cls,  # noqa
            v: List[CommandArgument],
    ) -> List[CommandArgument]:
        has_optional = False
        for arg in v:
            if has_optional and not arg.is_optional:
                raise ValueError('Required argument follows optional argument')

            has_optional = has_optional or arg.is_optional
        return v

class Module(BaseModel):
    name: str = Field(regex=r'[\w\d]+', min_length=1, max_length=255)
    description: Optional[str]
    url: AnyHttpUrl

    commands: Dict[str, Command]
    actions: List[str] = Field(default_factory=list)

    @validator('commands')
    def command_names_match(
            cls,  # noqa
            v: Dict[str, Command],
    ) -> Dict[str, Command]:
        if not all(key == command.name for key, command in v.items()):
            raise ValueError('Keys and command names must match')
        return v

    class Config:
        schema_extra = {
            'example': SAMPLE_MODULE
        }

class SlackMethod(str, Enum):
    ...

class SlackRequest(BaseModel):
    method: SlackMethod
    payload: Dict

log = logging.getLogger(__name__)

class ActionDispatcher:
    session: ClientSession
    storage: Storage

    def __init__(self, app: FastAPI) -> None:
```

```
        self.session = app.state.session
        self.storage = app.state.storage

    async def dispatch(self, payload: Dict[str, Any]) -> None:
        action_ids = set()

        if actions := payload.get('actions'):
            action_ids |= {
                f'{payload["type"]}:{action["action_id"]}'
                for action in actions
            }

        if action_callback_id := payload.get('callback_id'):
            action_ids.add(f'{payload["type"]}:{action_callback_id}')

        if view := payload.get('view'):
            if view_callback_id := view.get("callback_id"):
                action_ids.add(f'{payload["type"]}:{view_callback_id}')

        await self._trigger_all_actions(action_ids, payload)

    async def _trigger_all_actions(
            self,
            action_ids: Set[str],
            payload: Dict[str, Any],
    ) -> None:
        futures = []
        for action_id in action_ids:
            module = await self.storage.get_module_by_action(action_id)
            if module is not None:
                futures.append(
                    self._trigger_action(module, action_id, payload)
                )
        await asyncio.gather(*futures)

    async def _trigger_action(
            self,
            module: Module,
            action_id: str,
            metadata: Dict[str, Any]
    ) -> None:
        payload = {
            'metadata': metadata,
        }
        url = f'{module.url}/actions/{action_id}'
        async with self.session.post(url, json=payload) as resp:
            resp.raise_for_status()

class CommandDispatcher:
    session: ClientSession
    slack: WebClient
    storage: Storage

    def __init__(self, app: FastAPI) -> None:
        self.session = app.state.session
        self.slack = app.state.slack
        self.storage = app.state.storage

    async def dispatch(self, payload: Dict[str, str]) -> None:
        try:
            module, command, arguments = await self._parse_payload(payload)
        except ValueError as e:
            return await self._error(
                payload,
                str(e)
            )
```

```
    try:
        await self._trigger_command(module, command, arguments, payload)
    except ClientError:
        log.exception('Module request failed')
        await self._error(
            payload,
            'Command execution failed. '
            'Please consult with the administrator.'
        )

async def _parse_payload(
        self,
        payload: Dict[str, str],
) -> Tuple[Module, Command, List[str]]:
    parsed_text = split(payload['text'])
    if len(parsed_text) == 0:
        formatted_modules = self._format_strings(
            await self.storage.get_module_names()
        )
        raise ValueError(
            f'Usage: `{payload["command"]} [module] [command]`. '
            f'Available modules: {formatted_modules}'
        )
    elif len(parsed_text) == 1:
        parsed_text.append('')

    module_name, command_name, *arguments = parsed_text

    module = await self.storage.get_module(module_name)
    if module is None:
        formatted_modules = self._format_strings(
            await self.storage.get_module_names()
        )
        raise ValueError(
            f'Module `{module_name}` does not exist. '
            f'Available modules: {formatted_modules}'
        )

    try:
        command = module.commands[command_name]
    except KeyError:
        raise ValueError(
            f'Usage: `{payload["command"]} {module_name} [command]`. '
            f'Available commands: {self._format_strings(module.commands)}'
        )

    required_arguments = [x for x in command.arguments if not x.is_optional]
    if len(arguments) < len(required_arguments):
        arg_names = (arg.name for arg in required_arguments)
        raise ValueError(
            f'Missing one or more required arguments for {command_name}. '
            f'Required arguments: {self._format_strings(arg_names)}'
        )

    return module, command, arguments

async def _trigger_command(
        self,
        module: Module,
        command: Command,
        arguments: List[str],
        metadata: Dict[str, str],
) -> None:
    payload = {
        'arguments': {
```

```
            arg.name: value
            for arg, value in zip(command.arguments, arguments)
          },
          'metadata': metadata,
        }
        url = f'{module.url}/commands/{command.name}'
        async with self.session.post(url, json=payload) as resp:
          resp.raise_for_status()

    async def _error(self, payload: Dict[str, str], message: str) -> None:
        channel = payload['channel_id']
        user = payload['user_id']
        log.info(
          f'Error triggered by user {user} in channel {channel}. '
          f'Sending ephemeral error message: {message}'
        )
        await self.slack.chat_postEphemeral(
          channel=channel,
          user=user,
          text=message,
        )

    @staticmethod
    def _format_strings(strings: Iterable[str]) -> str:
        return ' '.join(f'`{x}`' for x in strings)

log = logging.getLogger(__name__)

async def get_storage(request: Request) -> 'Storage':
    return request.app.state.storage

class Storage:
    redis: Redis

    def __init__(self, redis: Redis) -> None:
        self.redis = redis

    @staticmethod
    def _encode_module(module: Module) -> str:
        return json.dumps(jsonable_encoder(module))

    @staticmethod
    def _decode_module(raw_module: str) -> Module:
        return Module.parse_raw(raw_module)

    async def add_or_replace_module(self, module: Module) -> None:
        tr = self.redis.multi_exec()
        tr.set(
          f'module:{module.name}',
          self._encode_module(module),
          expire=MODULE_EXPIRATION_SECONDS,
        )
        for action in module.actions:
          tr.set(
            f'action:{action}',
            module.name,
            expire=MODULE_EXPIRATION_SECONDS,
          )
        await tr.execute()

    async def get_module(self, module_name: str) -> Optional[Module]:
        raw_module = await self.redis.get(f'module:{module_name}')
        if raw_module is None:
          return None
        return self._decode_module(raw_module)
```

```
    async def get_module_by_action(self, action: str) -> Optional[Module]:
        module_name = await self.redis.get(f'action:{action}')
        if module_name is None:
            return None
        return await self.get_module(module_name.decode('utf-8'))

    async def _module_keys(self) -> AsyncIterator[str]:
        async for key in self.redis.iscan(match='module:*'):
            yield key.decode('utf-8')

    async def get_all_modules(self) -> Dict[str, Module]:
        results = {}
        async for key in self._module_keys():
            raw_module = await self.redis.get(key)
            if raw_module:
                module = self._decode_module(raw_module)
                results[module.name] = module
        return results

    async def get_module_names(self) -> List[str]:
        return [key.split(':')[-1] async for key in self._module_keys()]

log = logging.getLogger(__name__)

async def get_slack(request: Request) -> WebClient:
    return request.app.state.slack

async def slack_request(slack: WebClient, req: SlackRequest) -> SlackResponse:
    try:
        method = getattr(slack, req.method.value)
        return await method(**req.payload)
    except SlackApiError as e:
        log.exception('Slack request failed')
        raise HTTPException(e.response.status_code, e.response.get('error'))

router = APIRouter()

router.include_router(modules.router, prefix='/modules')
router.include_router(slack.router, prefix='/slack')

router = APIRouter()

class GetModulesResponse(BaseModel):
    modules: Dict[str, Module]

    class Config:
        schema_extra = {
            'example': {
                SAMPLE_MODULE['name']: SAMPLE_MODULE
            }
        }

@router.get('/', response_model=GetModulesResponse)
async def get_modules(
        storage: Storage = Depends(get_storage),
) -> GetModulesResponse:
    modules = await storage.get_all_modules()
    return GetModulesResponse(modules=modules)

@router.get('/{module_name}', response_model=Module)
async def get_module_by_name(
        module_name: str,
        storage: Storage = Depends(get_storage),
) -> Module:
    module = await storage.get_module(module_name)
```

```
    if module is None:
        raise HTTPException(status_code=404, detail="Module not found")

    return module

@router.post('/', response_model=Module)
async def register_module(
        module: Module,
        storage: Storage = Depends(get_storage),
) -> Module:
    await storage.add_or_replace_module(module)
    return module

router = APIRouter()

class SlackResponse(BaseModel):
    data: Dict

@router.post('/', response_model=SlackResponse)
async def request(
        req: SlackRequest,
        slack: WebClient = Depends(get_slack),
) -> SlackResponse:
    response = await slack_request(slack, req)
    return SlackResponse(data=response.data)

app = FastAPI(title=APP_TITLE)

app.add_event_handler('startup', start_app_handler(app))
app.add_event_handler('shutdown', stop_app_handler(app))

for route in slackers_router.routes:
    route.include_in_schema = False

app.include_router(slackers_router, prefix=SLACKERS_PREFIX, tags=['slack'])
app.include_router(api_router, prefix=API_PREFIX, tags=['metabot'])

class CommandMetadata(BaseModel):
    token: str
    command: str
    response_url: str
    trigger_id: str
    user_id: str
    user_name: str
    channel_id: str
    text: str

class CommandPayload(BaseModel):
    arguments: Dict[str, str]
    metadata: CommandMetadata

class ActionMetadata(BaseModel):
    type: str  # noqa
    token: Optional[str]
    team: Optional[Dict]
    user: Optional[Dict]
    response_url: Optional[str]
    actions: Optional[List[Dict]]
    api_app_id: Optional[str]
    callback_id: Optional[str]
    channel: Optional[Dict]
    container: Optional[Dict]
    hash: Optional[str]  # noqa
    is_cleared: Optional[bool]
    message: Optional[Dict]
    trigger_id: Optional[str]
```

```
    view: Optional[Dict]

class ActionPayload(BaseModel):
    metadata: ActionMetadata

command_metadata: ContextVar[Optional['CommandMetadata']] = ContextVar(
    'command_metadata',
    default=None
)

action_metadata: ContextVar[Optional['ActionMetadata']] = ContextVar(
    'action_metadata',
    default=None
)

current_module: ContextVar[Optional['Module']] = ContextVar(
    'current_module',
    default=None
)

def get_current_user_id() -> Optional[str]:
    if c := command_metadata.get():
        return c.user_id
    elif a := action_metadata.get():
        user = a.user or {}
        return user.get('id')
    return None

def get_current_channel_id() -> Optional[str]:
    if c := command_metadata.get():
        return c.channel_id
    elif a := action_metadata.get():
        channel = a.channel or {}
        return channel.get('id')
    return None

def slack_request(method: str, payload: Dict) -> Dict:
    module = current_module.get()
    assert module is not None, 'Must be called from any Slack context'

    api = SyncApis(module.metabot_client).metabot_api
    resp = api.request_api_slack_post(
        SlackRequest(
            method=method,
            payload=payload
        )
    )
    return resp.data

async def async_slack_request(method: str, payload: Dict) -> Dict:
    module = current_module.get()
    assert module is not None, 'Must be called from any Slack context'

    api = AsyncApis(module.metabot_client).metabot_api
    resp = await api.request_api_slack_post(
        SlackRequest(
            method=method,
            payload=payload
        )
    )
    return resp.data

    AsyncApis,
    ApiClient,
    models
)
```

```
log = logging.getLogger(__name__)

Converter = Callable[[str], Any]

@dataclass
class Argument:
    name: str
    is_optional: bool
    type: Type = str  # noqa A003
    description: Optional[str] = None

@dataclass
class Command:
    name: str
    func: Callable
    description: Optional[str] = None
    arguments: List[Argument] = field(default_factory=list)

class Module:
    name: str
    description: Optional[str]
    module_url: str
    metabot_client: ApiClient
    heartbeat_delay: float

    _commands: Dict[str, Command]
    _views: Dict[str, Callable]
    _actions: Dict[str, Callable]
    _converters: Dict[Type, Converter]
    _heartbeat: Optional[Task]

    def __init__(
            self,
            name: str,
            module_url: str,
            metabot_url: str,
            description: Optional[str] = None,
            heartbeat_delay: float = 10,
    ) -> None:
        self.name = name
        self.description = description
        self.module_url = module_url
        self.metabot_client = ApiClient(host=metabot_url)
        self.heartbeat_delay = heartbeat_delay

        self._commands = {}
        self._views = {}
        self._actions = {}
        self._converters = {}
        self._heartbeat = None

    def command(
            self,
            name: str,
            *,
            function: Optional[Callable] = None,
            description: Optional[str] = None,
            arg_descriptions: Optional[Dict[str, str]] = None,
    ) -> Callable:
        assert name not in self._commands, 'Duplicate command names detected'

        def wrapper(f: Callable) -> Callable:
            arguments = self._parse_arguments(f, arg_descriptions)
            self._commands[name] = Command(
                name=name,
```

```
            func=f,
            description=description,
            arguments=arguments
        )
        return f

    if function is None:
        return wrapper
    else:
        return wrapper(function)

def converter(
        self,
        to_type: Type,
        *,
        converter: Optional[Converter] = None,
) -> Callable:
    assert to_type not in self._converters, 'Duplicate converters detected'

    def wrapper(f: Callable) -> Callable:
        self._converters[to_type] = f
        return f

    if converter is None:
        return wrapper
    else:
        return wrapper(converter)

def action(
        self,
        name: str,
        *,
        function: Optional[Callable] = None,
) -> Callable:
    assert name not in self._actions, 'Duplicate action names detected'

    def wrapper(f: Callable) -> Callable:
        self._actions[name] = f
        return f

    if function is None:
        return wrapper
    else:
        return wrapper(function)

def view(
        self,
        name: str,
        *,
        function: Optional[Callable] = None
) -> Callable:
    assert name not in self._views, 'Duplicate view names detected'

    def wrapper(f: Callable) -> Callable:
        self._views[name] = f
        return f

    if function is None:
        return wrapper
    else:
        return wrapper(function)

@staticmethod
def _parse_arguments(
        f: Callable,
        arg_descriptions: Optional[Dict[str, str]] = None,
```

```
) -> List[Argument]:
    sig = signature(f)
    params = sig.parameters

    allowed_kind = (Parameter.POSITIONAL_OR_KEYWORD, Parameter.KEYWORD_ONLY)
    assert all(param.kind in allowed_kind for param in params.values()), (
        'Positional-only arguments, *args and **kwargs are not allowed'
    )

    arg_descriptions = arg_descriptions or {}
    return [Argument(
        name=p.name,
        is_optional=p.default is not Parameter.empty,
        type=p.annotation if p.annotation is not Parameter.empty else str,
        description=arg_descriptions.get(p.name),
    ) for p in params.values()]

async def execute_command(
        self,
        name: str,
        arguments: Dict[str, str],
) -> None:
    command = self._commands[name]
    converted_args = {
        arg.name: await self._maybe_await(
            self._get_converter(arg.type),
            value
        )
        for arg in command.arguments
        if (value := arguments.get(arg.name))
    }
    await self._maybe_await(command.func, **converted_args)

async def execute_action(self, action_id: str) -> None:
    action_type, action_name = action_id.split(':', 2)
    if action_type == 'block_actions':
        await self._maybe_await(self._actions[action_name])
    elif action_type == 'view_submission':
        await self._maybe_await(self._views[action_name])
    else:
        log.error(f'Unknown action {action_id} triggered')

@staticmethod
async def _maybe_await(
        function: Callable,
        *args: Any,
        **kwargs: Any
) -> Any:
    if iscoroutinefunction(function):
        return await function(*args, **kwargs)
    else:
        return function(*args, **kwargs)

def _get_converter(self, to_type: Type) -> Converter:
    return self._converters.get(to_type, to_type)

def install(self, app: FastAPI, prefix: str = '') -> FastAPI:
    async def set_current_module() -> None:
        current_module.set(self)

    app.include_router(
        router,
        prefix=prefix,
        tags=['metabot', self.name],
        dependencies=[Depends(set_current_module)],
    )
```

```
        if self.heartbeat_delay:
            app.add_event_handler('startup', self._start_heartbeat)
            app.add_event_handler('shutdown', self._stop_heartbeat)

        return app

    def _start_heartbeat(self) -> None:
        module = self._build_module_payload()
        metabot_api = AsyncApis(self.metabot_client).metabot_api

        async def heartbeat() -> None:
            while True:
                try:
                    await metabot_api.register_module_api_modules_post(module)
                except ApiException:
                    log.exception('Heartbeat to metabot server has failed')

                await asyncio.sleep(self.heartbeat_delay)

        self._heartbeat = asyncio.create_task(heartbeat())

    def _build_module_payload(self) -> models.Module:
        return models.Module(
            name=self.name,
            description=self.description,
            url=self.module_url,
            commands={
                command.name: models.Command(
                    name=command.name,
                    description=command.description,
                    arguments=[
                        models.CommandArgument(
                            name=arg.name,
                            description=arg.description,
                            is_optional=arg.is_optional,
                        ) for arg in command.arguments
                    ],
                ) for command in self._commands.values()
            },
            actions=(
                [f'block_actions:{action}' for action in self._actions]
                + [f'view_submission:{view}' for view in self._views]
            )
        )

    def _stop_heartbeat(self) -> None:
        if self._heartbeat is not None:
            self._heartbeat.cancel()
            self._heartbeat = None

    current_module,
    command_metadata,
    action_metadata,
)

router = APIRouter()

@router.post('/commands/{command_name}')
async def execute_command(
        command_name: str,
        payload: CommandPayload,
        background_tasks: BackgroundTasks,
) -> None:
    module = current_module.get()
    if module is None:
```

```
        raise HTTPException(500)

    command_metadata.set(payload.metadata)

    background_tasks.add_task(
        module.execute_command,
        command_name,
        jsonable_encoder(payload.arguments),
    )

@router.post('/actions/{action_id}')
async def execute_action(
        action_id: str,
        payload: ActionPayload,
        background_tasks: BackgroundTasks,
) -> None:
    module = current_module.get()
    if module is None:
        raise HTTPException(500)

    action_metadata.set(payload.metadata)
    background_tasks.add_task(module.execute_action, action_id)

app = FastAPI()
app.add_event_handler('startup', start_app_handler(app))
app.add_event_handler('shutdown', stop_app_handler(app))

module = Module(
    name='vacations',
    description=':palm_tree: Manage vacations, days off & other leaves',
    module_url=MODULE_URL,
    metabot_url=METABOT_URL,
    heartbeat_delay=HEARTBEAT_DELAY
)

class UserId(str):
    pass

@module.converter(UserId)
async def convert_user_id(user_id: str) -> UserId:
    user_id_search = search(r'<@(\w+)\|(\w+)>', user_id)

    if user_id_search:
        return UserId(user_id_search.group(1))
    else:
        await send_ephemeral(f'Invalid user mention: `{user_id}`')
        raise ValueError('Invalid UserId')

@module.converter(date)
async def convert_date(iso_date: str) -> date:
    try:
        return date.fromisoformat(iso_date)
    except ValueError as e:
        await send_ephemeral(f'Invalid date format: `{iso_date}`')
        raise e

@module.converter(Decimal)
async def convert_decimal(number: str) -> Decimal:
    try:
        return Decimal(number)
    except ArithmeticError as e:
        await send_ephemeral(f'Invalid number: `{number}`')
        raise e

@module.command(
    'request',
```

```
    description='Request a vacation, day off or another leave.\n'
                'If no arguments are provided, opens a dialog with a form to '
                'fill out your request.',
    arg_descriptions={
        'leave_type': f'Type of your leave '
                      f'(one of `{"` `".join(LEAVE_TYPES)}`)',
        'date_from': 'The day you want to start your leave (e.g. `2020-10-29`)',
        'date_to': 'End date of your leave (e.g. `2020-10-31`)',
        'reason': 'Reason for your leave '
                  '(e.g. `"Going on a trip with my family"`)'
    }
)
async def request(
        leave_type: str = None,
        date_from: date = None,
        date_to: date = None,
        reason: str = '',
) -> None:
    if leave_type is None:
        return await open_request_view(app.state.users)

    await create_request(
        app.state.history,
        app.state.users,
        leave_type,
        date_from,
        date_to,
        reason
    )

@module.command(
    'approve',
    description='Approve a vacation request.\n'
                'Available only in the admin channel.',
    arg_descriptions={
        'request_id': 'Id of the request to be approved '
                      '(e.g. `5eb95d610ec29ce040c8c144`)'
    }
)
async def approve(request_id: str) -> None:
    if not await is_admin_channel():
        return await send_ephemeral(
            'This command is available only in the admin channel.'
        )

    await approve_request(
        app.state.history,
        app.state.users,
        request_id,
    )

@module.command(
    'deny',
    description='Deny a vacation request.\n'
                'Available only in the admin channel.',
    arg_descriptions={
        'request_id': 'Id of the request to be denied '
                      '(e.g. `5eb95d610ec29ce040c8c144`)'
    }
)
async def deny(request_id: str) -> None:
    if not await is_admin_channel():
        return await send_ephemeral(
            'This command is available only in the admin channel.'
        )
```

```
    await deny_request(app.state.history, request_id)

@module.command(
    'stats',
    description='View available leave days and leaves history of a user '
                '(or yourself if no user is provided).\n'
                'Stats of users other than you are only available '
                'in the admin channel.',
    arg_descriptions={
        'user': 'Mention of a user (e.g. `<@metabot>`)'
    }
)
async def stats(user: UserId = None) -> None:
    await send_history(app.state.history, app.state.users, user)

@module.command(
    'add',
    description='Add more leave days to a user (or all users). '
                'Only available in the admin channel.',
    arg_descriptions={
        'leave_type': f'Type of leave you want to add days for '
                      f'(one of `{"` `".join(LEAVE_TYPES)}`)',
        'days': 'Number of days (e.g. `5`)',
        'user': 'Mention of a user (e.g. `<@metabot>`)'
    }
)
async def add(leave_type: str, days: Decimal, user: UserId = None) -> None:
    if not await is_admin_channel():
        return await send_ephemeral(
            'This command is available only in the admin channel.'
        )

    await add_days(app.state.users, leave_type, days, user)

@module.view(REQUEST_VIEW_ID)
async def request_view() -> None:
    leave_type, date_from, date_to, reason = parse_request_view()
    await create_request(
        app.state.history,
        app.state.users,
        leave_type,
        date_from,
        date_to,
        reason
    )

@module.action(APPROVE_BUTTON_ACTION_ID)
async def approve_button_action() -> None:
    if not await is_admin_channel():
        return await send_ephemeral(
            'This command is available only in the admin channel.'
        )

    request_id = await get_request_id_from_button()
    await approve_request(
        app.state.history,
        app.state.users,
        request_id,
    )

@module.action(DENY_BUTTON_ACTION_ID)
async def deny_button_action() -> None:
    if not await is_admin_channel():
        return await send_ephemeral(
            'This command is available only in the admin channel.'
        )
```

```
    request_id = await get_request_id_from_button()
    await deny_request(app.state.history, request_id)

module.install(app)

app = FastAPI()
app.add_event_handler('startup', start_app_handler(app))
app.add_event_handler('shutdown', stop_app_handler(app))

module = Module(
    name='feedback',
    description=':ledger: Create questionnaires and collect feedback',
    module_url=MODULE_URL,
    metabot_url=METABOT_URL,
    heartbeat_delay=HEARTBEAT_DELAY
)

@module.command(
    'create',
    description='Create a questionnaire & send it out to others.\n'
                'Opens a dialog with a form to create the questions',
    arg_descriptions={
        'num_of_questions': f'Number of questions you want to ask '
                            f'(max: {MAX_QUESTIONS})'
    }
)
async def create(num_of_questions: int = 1) -> None:
    if not 1 <= num_of_questions <= MAX_QUESTIONS:
        return await send_ephemeral(
            f'Number of questions must be between 1 and {MAX_QUESTIONS}'
        )

    await open_creation_view(num_of_questions)

@module.view(CREATION_VIEW_ID)
async def creation_view() -> None:
    title, recipients, questions = parse_creation_view()
    await create_questionnaire(
        app.state.feedback,
        title,
        recipients,
        questions
    )

@module.view(ANSWER_VIEW_ID)
async def answer_view() -> None:
    q_id, answers = parse_answer_view()
    await submit_answers(
        app.state.feedback,
        q_id,
        answers
    )

@module.action(NOTIFY_ACTION_ID)
async def notify() -> None:
    q_id = await get_q_id_from_button()
    await notify_recipients(app.state.feedback, q_id)

@module.action(ANSWER_ACTION_ID)
async def answer() -> None:
    q_id = await get_q_id_from_button()
    await open_answer_view(app.state.feedback, q_id)

module.install(app)
```

```
log = logging.getLogger(__name__)
app = FastAPI()

module = Module(
    name='help',
    description=':sos: Get info about installed MetaBot modules and commands',
    module_url=MODULE_URL,
    metabot_url=METABOT_URL,
    heartbeat_delay=HEARTBEAT_DELAY,
)

@module.command(
    'me',
    description='Displays info about a module and lists available commands.\n'
                'When module name is not provided, displays short info about '
                'all modules.',
    arg_descriptions={
        'module_name': 'Module name',
    }
)
async def get_help(module_name: str = None) -> None:
    await send_help(module.metabot_client, module_name)

@module.action(COMMANDS_BUTTON_ACTION_ID)
async def module_help_action() -> None:
    module_name = await get_module_name_from_button()
    await send_help(module.metabot_client, module_name)

module.install(app)
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

КОРПОРАТИВНИЙ БОТ ДЛЯ ІТ-КОМПАНІЙ

Програма та методика тестування

КПІ.ІП-6305.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ А.А. Коротенко

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.В. Васюк

Київ – 2020 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Корпоративний бот для IT-компаній, що являє собою модульний чатбот для Slack, створений на мові Python з використанням фреймворку FastAPI.

2 МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

– функціональна працездатність команд та інтерактивних елементів;
– правильність роботи кожного з модулів бота;
– забезпечення належного рівня безпеки даних;
– зручність роботи з ботом;
– відповідність вимогам Технічного завдання.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

– функціональне тестування, зокрема на рівні Critical path test (базове тестування);

– тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);

– тестування інтерфейсу.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію Katalon Studio.

Працездатність бота перевіряється шляхом:

– динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;

– динамічного ручного тестування на відповідність функціональним вимогам;

– статичного тестування коду;

– тестування бота на різних платформах;

– тестування при максимальному навантаженні;

– тестування стабільності роботи при різних умовах;

– тестування зручності використання;

– тестування інтерфейсу.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

КОРПОРАТИВНИЙ БОТ ДЛЯ ІТ-КОМПАНІЙ

Керівництво користувача

КПІ.ІП-6305.045490.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ А.А. Коротенко

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.В. Васюк

Київ – 2020 року

Для початку роботи з ботом, потрібно встановити офіційний додаток Slack на будь-який пристрій та отримати запрошення в команду, де вже є встановлений бот.

Точка входу для взаємодії з ботом – введення в чат будь-якого каналу Slack команди “**/meta <назва модулю> <назва команди> <аргументи>**”. Доступні наступні модулі:

– **help** (довідка про модулі);

– **vacations** (модуль оформлення відпусток);

– **feedback** (модуль проведення опитувань).

Модуль help. Цей вбудований модуль містить єдину команду – “**/meta help me**”, що виводить на екран інформацію про всі активні модулі бота. Поруч з інформацією про кожен модуль знаходиться кнопка “**Commands**”, після натискання на яку на екран виводиться повний опис всіх команд даного модулю. Приклад роботи з даною командою зображено на рисунку 1.

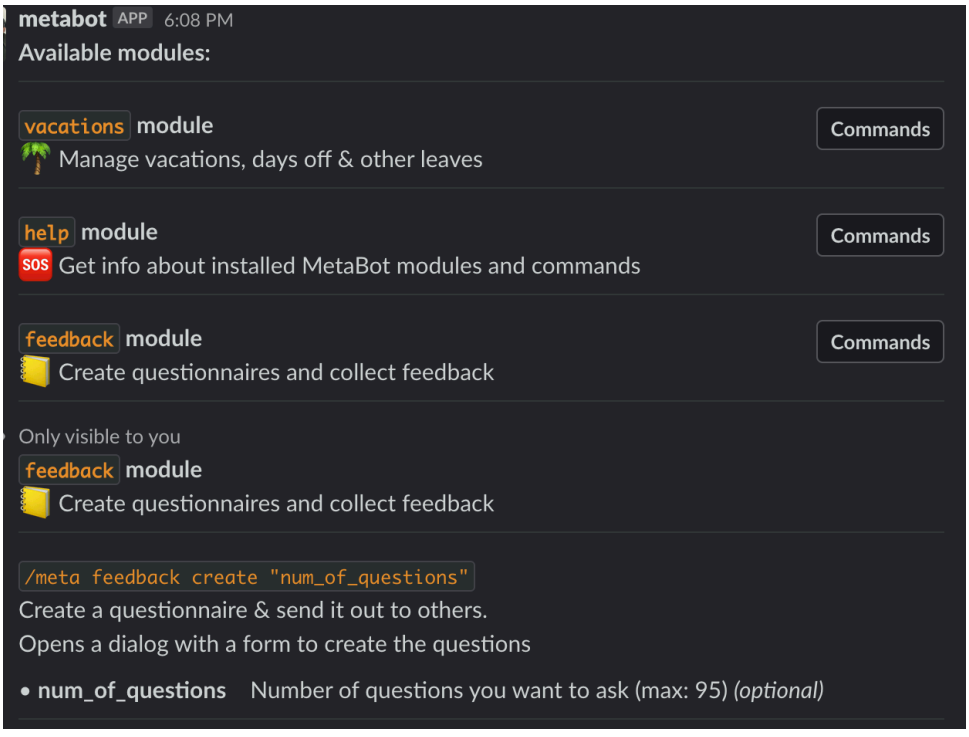


Рисунок 1 – Приклад роботи команди **/meta help me**

Модуль vacations. Для початку роботи з даним модулем потрібно ввести команду “**/meta vacations request**” – ця команда створює запит на вихід у відпустку. Якщо не введено жодного аргументу, бот відкриє в Slack форму для заповнення, яку зображено на рисунку 2.

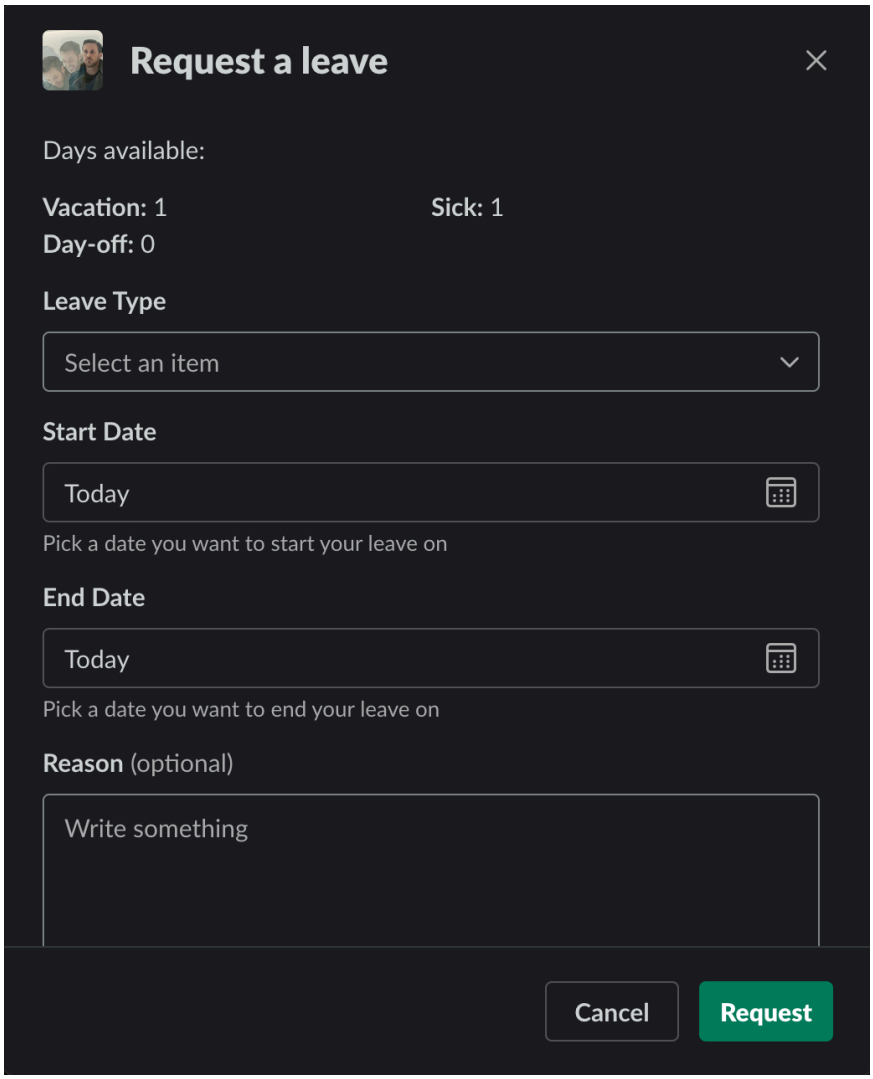


Рисунок 2 – Форма створення запиту на відпустку

Інший спосіб – заповнення всіх параметрів запиту в тексті команди, наприклад “/meta vacations request sick 2020-05-31 2020-06-20 коронавірус”.

Перший аргумент команди – тип відпустки:

– **vacation** (звичайна відпустка);

– **sick** (лікарняний);

– **day-off** (відгул).

Кількість доступних по кожному з типів відпусток, а також власну історію попередніх відпусток можна переглянути за допомогою команди “**/meta vacations stats**”. Вигляд результату виконання команди зображено на рисунку 3. Доступні дні нараховуються кожного дня згідно налаштувань сервера.

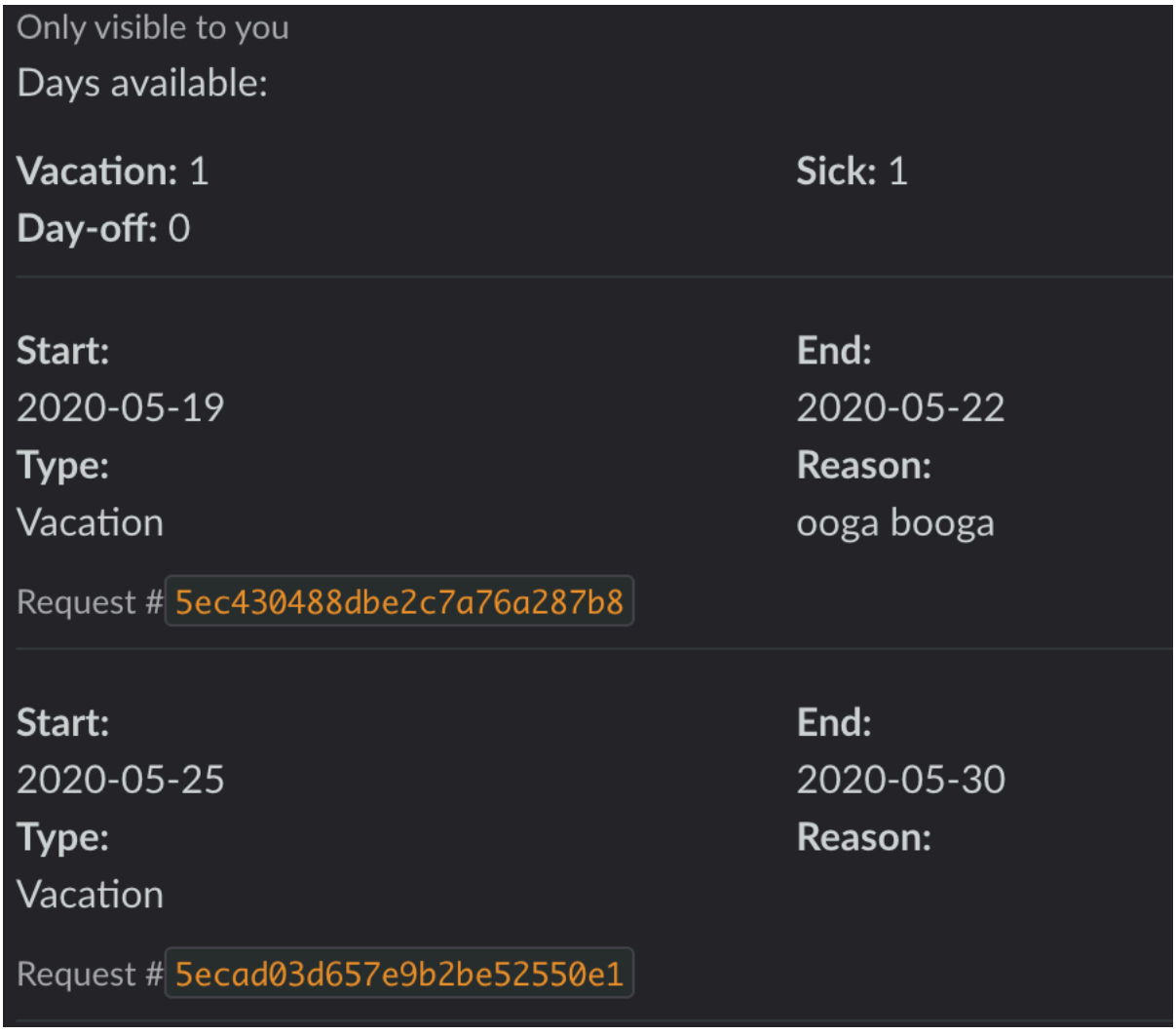


Рисунок 3 – Статистика відпусток користувача

Другий та третій аргумент – період, на який створюється запит. Обидві дати потрібно ввести у форматі ISO (YYYY-MM-DD). Дати повинні бути пізніше дати створення запиту, інакше бот повідомить користувача про помилку.

Четвертий аргумент – причина запиту (опціонально).

В обох випадках вдалого створення запиту (через форму або команду), бот надішле повідомлення наступного вигляду: *«Leave request #5ed3cd88657e9b2be52550e2 has been created! You will be notified when your request is approved or denied.»*, а також відправить в закритий канал адміністраторів повідомлення, зображене на рисунку 4.

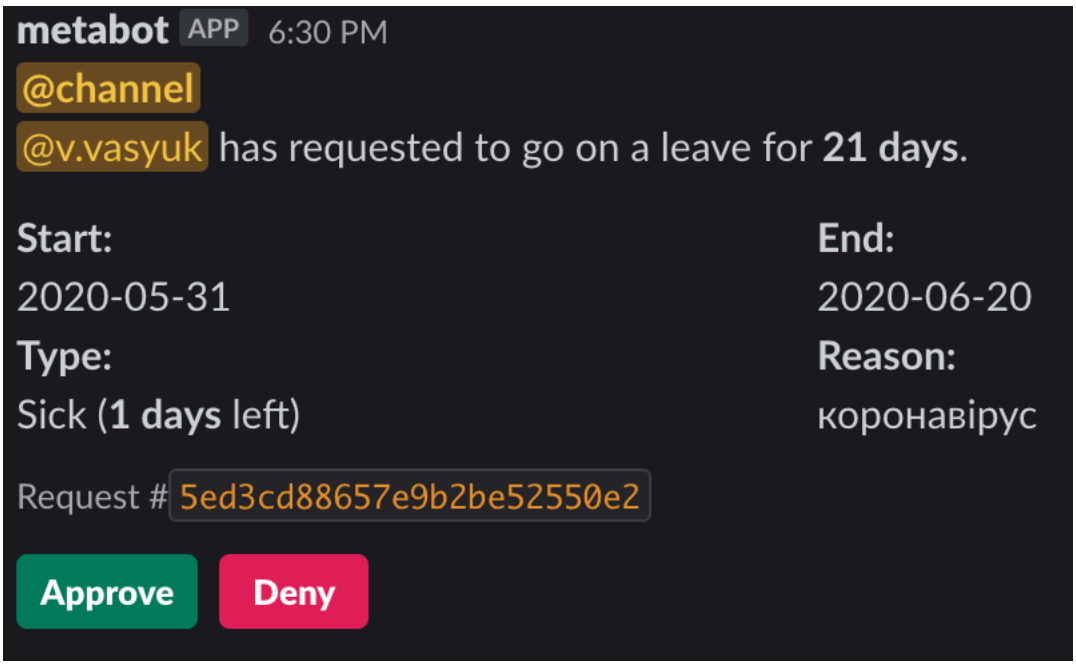


Рисунок 4 – Запит на відпустку в каналі адміністраторів

Канал адміністраторів вказаний при налаштуванні сервера бота, і всі користувачі з доступом до цього каналу зможуть прийняти чи відхилити запит. Рекомендується надати доступ до цього каналу працівникам компанії, що будуть відповідати за прийняття відпусток (менеджери, відділ кадрів тощо).

В разі відхилення запиту (натискання на кнопку “Deny”), кількість доступних днів не зміниться, а бот надішле повідомлення наступного вигляду: *«❌Your leave request #5ed3cd88657e9b2be52550e2 has been denied by @v.vasyuk.»*

В разі схвалення запиту (натискання на кнопку “Approve”), кількість доступних днів зменьшиться, а бот надішле повідомлення наступного вигляду:

«✓ Your leave request #5ed3cd88657e9b2be52550e2 has been approved by @v.vasyuk.»

Додатково, модуль vacations надсилає в канал **#announcements** (назву каналу можна змінити в налаштуваннях) повідомлення кожного дня, якщо хтось з користувачів в цей день у відпустці. Для використання цього функціоналу достатньо лише зайти у канал. Вигляд повідомлення про користувачів у відпустці зображено на рисунку 5.

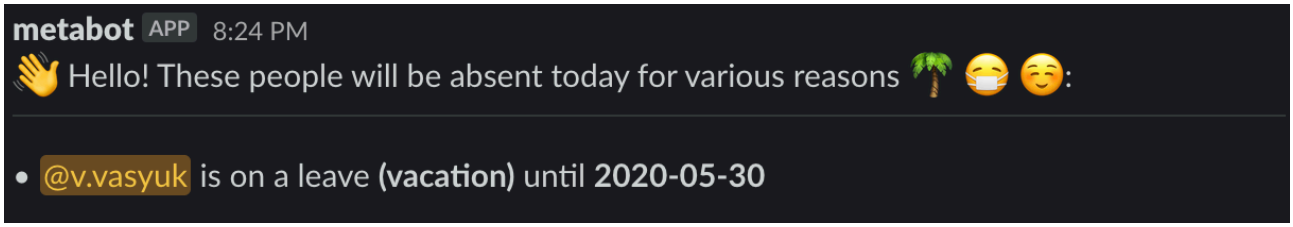


Рисунок 5 – Щоденне повідомлення модуля відпусток

Повний перелік команд модуля vacations можна переглянути за допомогою команди “**/meta help me vacations**”.

Модуль feedback. Для початку роботи з даним модулем потрібно ввести команду “**/meta feedback create**” – ця команда відкриває форму для створення опитувальника, зображену на рисунку 6. Команда приймає єдиний аргумент – кількість запитань в опитуванні, тобто при введенні команди “/meta feedback create 5” відкриється форма для створення опитувальника з 5 питаннями. Максимальна кількість запитань – 95 (через обмеження в розмірах форм Slack).

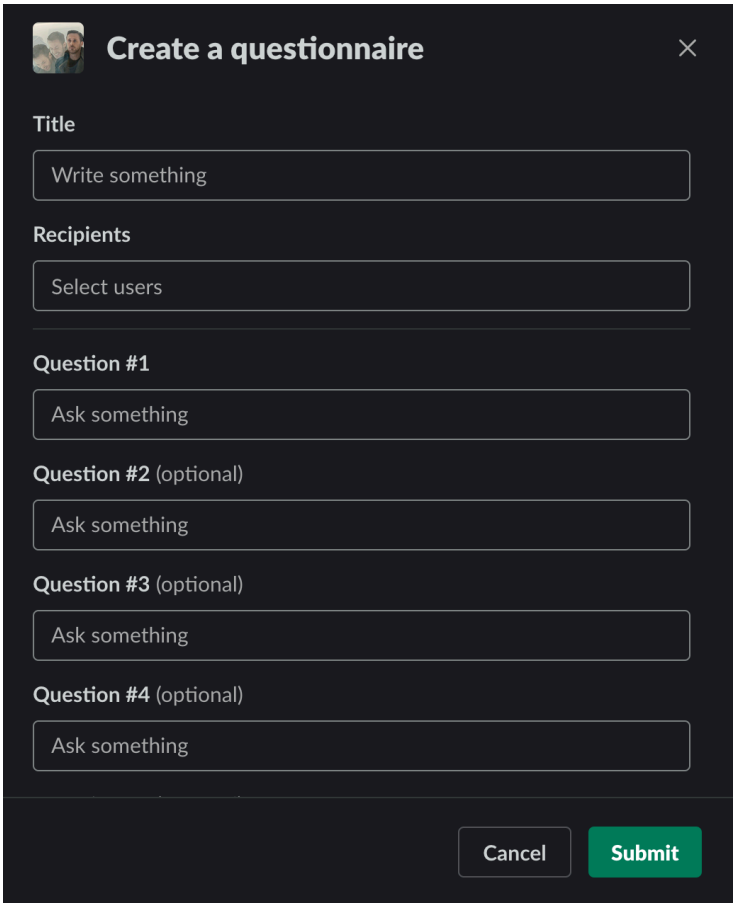


Рисунок 6 – Форма створення опитувальника

В формі потрібно ввести назву опитувальника, обрати отримувачів та ввести всі запитання.

Після створення опитування, автор отримує спеціальне повідомлення, зображене на рисунку 7. Всі відповіді від учасників бот збирає у відповідях до спеціального повідомлення. Також, автор може натиснути на кнопку “**Notify**” для повторної відправки опитування учасникам, які ще не надали відповіді.

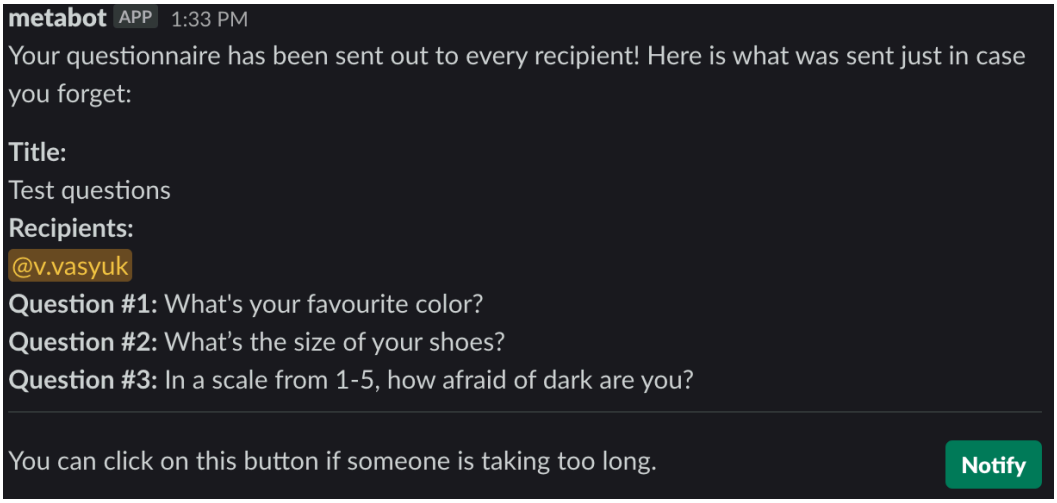


Рисунок 7 – Повідомлення після створення опитування

Всі учасники отримують повідомлення вигляду *«@v.vasyuk is asking for your feedback on "Test questions"! Click on the button to answer some questions.»*. При натисканні на кнопку «Answer» в цьому повідомленні, відкривається форма для заповнення відповідей, зображена на рисунку 8.

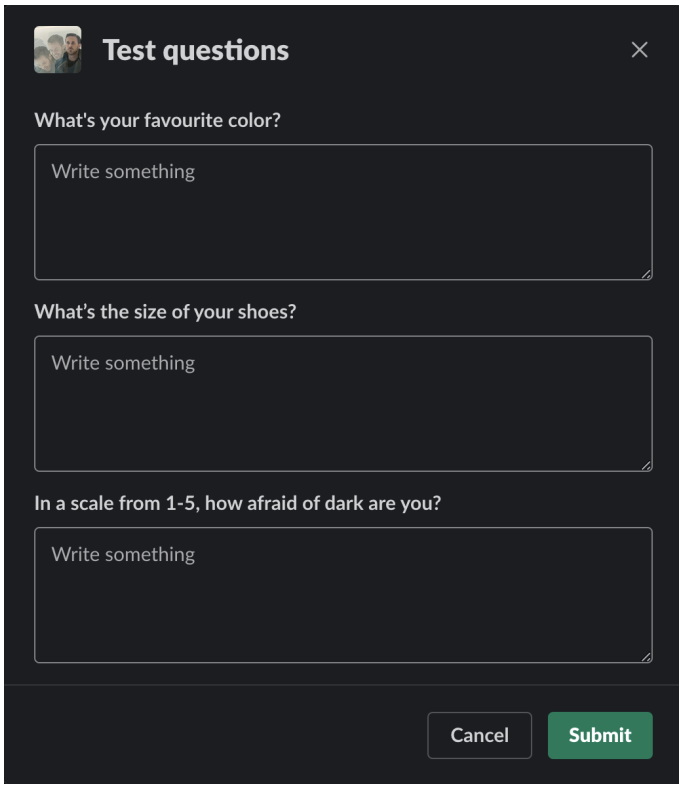


Рисунок 8 – Форма відповіді на опитування

Як вже зазначалося вище, всі відповіді учаників приходять автору у відповіді до початкового повідомлення, та мають вигляд, зображений на рисунку 9.

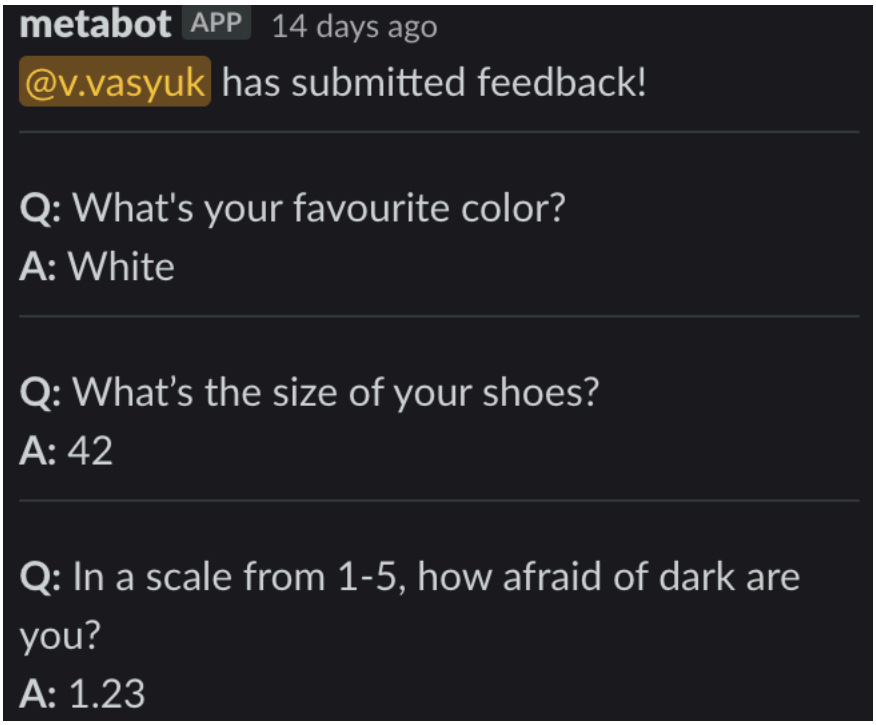


Рисунок 9 – Вигляд повідомлення з відповідями учасника

В разі завершення опитування, автор отримує сповіщення вигляду «All recipients have sent their answers! 🎉».

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

КОРПОРАТИВНИЙ БОТ ДЛЯ ІТ-КОМПАНІЙ

Керівництво програміста

КПІ.ІП-6305.045490.06.33

“ПОГОДЖЕНО”

Керівник проєкту:

_____ А.А. Коротенко

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.В. Васюк

Київ – 2020 року

Для розробки нового модуля бота спочатку потрібно підготовити робоче середовище:

– створити Slack-додаток за посиланням https://api.slack.com/apps;

– клонувати репозиторій Github за посиланням https://github.com/vasyukvv42/metabot;

– встановити та налаштувати Docker і docker-compose;

– відкрити порт 8000 на локальній машині якщо наявна зовнішня IP-адреса, або скористатися port-forward на віддалений сервер (ssh-tunnel, telepresence, ngrok тощо);

– запустити головний сервер бота за допомогою команди "**docker-compose up --build metabot**";

– перейти в налаштування Slack-додатку;

– перейти на вкладку Slash Commands, створити команду /meta, вказати адресу сервера зі шляхом /slack/commands;

– перейти на вкладку Interactivity & Shortcuts, вказати адресу сервера зі шляхом /slack/actions.

Після цього команда "**/meta**" стане доступна в Slack.

Перед розробкою модулів також варто детально ознайомитися з архітектурою та принципом роботи бота, наведених в репозиторії Github та в другому розділі пояснювальної записки.

Оскільки для створення нового модуля потрібно реалізувати звичайний HTTP-сервер з декількома методами API, для цього можна застосувати будь-яку високорівневу мову програмування з підтримкою мережевих протоколів. Однак для спрощення та пришвидшення процесу, було створено допоміжну бібліотеку **fastapi-metabot** на мові Python, яка створює потрібні методи API та реалізує механізм heartbeat автоматично, і має зручний інтерфейс додавання нових команд чи інтерактивних подій в модуль. Аналогічні бібліотеки за потреби можуть бути створені в майбутньому і для інших популярних мов програмування.

Для встановлення бібліотеки потрібно використати команду “**pip install fastapi-metabot**”. Після чого, в основному Python-файлі потрібно створити об’єкт класу fastapi_metabot.module.Module, як показано на рисунку 1.

```python
from fastapi import FastAPI
from fastapi_metabot.module import Module

app = FastAPI()

module = Module(
    name='example',
    description='Example module',
    module_url='http://example:8000',  # URL of this module
    metabot_url='http://metabot:8000'  # URL of the Metabot server
)

# this creates a few routes for Metabot and starts the heartbeat
module.install(app)
```

Рисунок 1 – Приклад створення модуля

Після чого для запуску модуля слід запустити команду “**uvicorn main:app**”. В даному прикладі, буде створено модуль “**example**” без команд та інтерактивних подій. Зверніть увагу, що в параметрі “**metabot_url**” потрібно вказати URL головного сервера бота, а в параметрі “**module_url**” – URL поточного модуля, що має бути досяжним з боку головного сервера. Рекомендується зчитувати дані параметри зі змінних оточення чи конфігураційних файлів, оскільки URL модуля та головного сервера можуть відрізнятися в різних середовищах.

Для додавання команд (slash command) в модуль, потрібно скористатись методом-декоратором **command()**, як показано на рисунку 2.

```python
# /meta example test "param" "optional"
# command arguments are based on your function parameters
@module.command('test')
async def test(param: int, optional: str = 'default') -> None:
    # argument values are casted from str according to the annotation
    # use @module.converter(Type) to provide custom cast functions
    print(type(param))  # <class 'int'>

    # you can access any Slack API method via the client
    await async_slack_request(
        method='chat_postMessage',
        payload={
            'text': 'Hello, World!',
            'channel': '#general'
        }
    )

    # Slack command/action payloads are stored inside contextvars
    payload = command_metadata.get()
    print(payload.user_name)  # current user's name
```

Рисунок 2 – Приклад створення команди

В даному прикладі в боті з'явиться команда "**/meta example test**", яка приймає два аргументи – "**param**" (обов'язковий аргумент, автоматично конвертується в тип int) та "**optional**" (опціональний аргумент типу str). Конвертація тексту, введеного користувачем як значення аргументів, відбувається автоматично згідно анотацій типів; додатково можна вказати свої функції для конвертації в потрібний тип за допомогою методу **converter()**.

В прикладі також показано, як можна за допомогою методу **fastapi_metabot.utils.async_slack_request** здійснити запит до Slack API. Повний перелік доступних методів можна знайти в документації API головного сервера за посиланням **/docs**, а опис параметрів кожного методу – в офіційній документації Slack.

Контекст поточної команди (дані, що надходять з боку Slack) можна отримати за допомогою контекстної змінної

fastapi_metabot.utils.command_metadata. З детальним описом всіх параметрів, що надсилає Slack під час виконання команди, можна ознайомитися в офіційній документації Slack API. Також доступні функції для отримання ID поточного користувача Slack (**fastapi_metabot.utils.get_current_user_id()**) та ID поточного каналу Slack (**fastapi_metabot.utils.get_current_channel_id()**).

За допомогою бібліотеки також можна реагувати на інтерактивні події (метод action) та на подію відправки користувачем форми (метод view). Приклад додавання подій та форм зображено на рисунку 3.

```
@module.action('action_id')
async def action() -> None:
    # this is called when someone triggers an interactive component
    payload = action_metadata.get()
    print(payload.actions)


@module.view('callback_id')
async def view() -> None:
    # this is called when someone submits a modal
    payload = action_metadata.get()
    print(payload.view)
```

Рисунок 3 – Приклад додавання подій та форм

В даному прикладі, модуль буде викликати вказані функції, коли користувачі виконують подію з ідентифікатором “**action_id**” або відправляють форму з ідентифікатором “**callback_id**”. Зверніть увагу, що ідентифікатори подій мають бути унікальними серед всіх модулів.

Аналогічно до контексту команд, повну інформацію про поточну подію чи форму можна отримати за допомогою контекстної змінної **fastapi_metabot.utils.action_metadata**. З детальним описом всіх параметрів, що надсилає Slack, можна ознайомитися в офіційній документації Slack API.

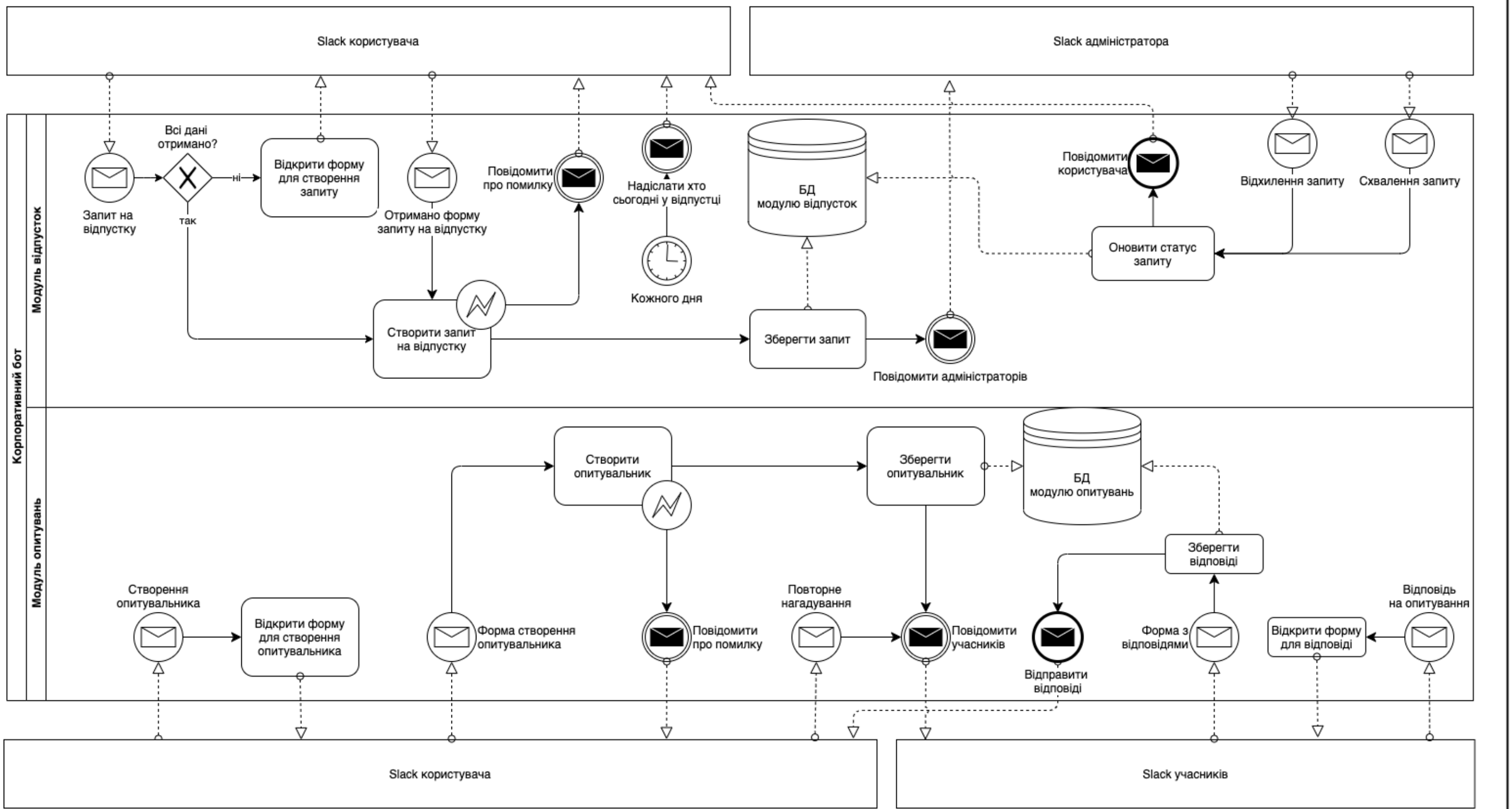
Після завершення написання модулю та тестового покриття, для розгортання модуль потрібно контейнеризувати за допомогою Docker та додати в файл “**docker-compose.yml**”. Запис в цьому файлі на прикладі вбудованого модулю help зображено на рисунку 4.

```
38    help:
39      build:
40        context: modules/help
41        dockerfile: Dockerfile
42        target: dev
43      restart: unless-stopped
44      volumes:
45      - './modules/help:/app'
46      depends_on:
47        - metabot
48      env_file: modules/help/.env
```

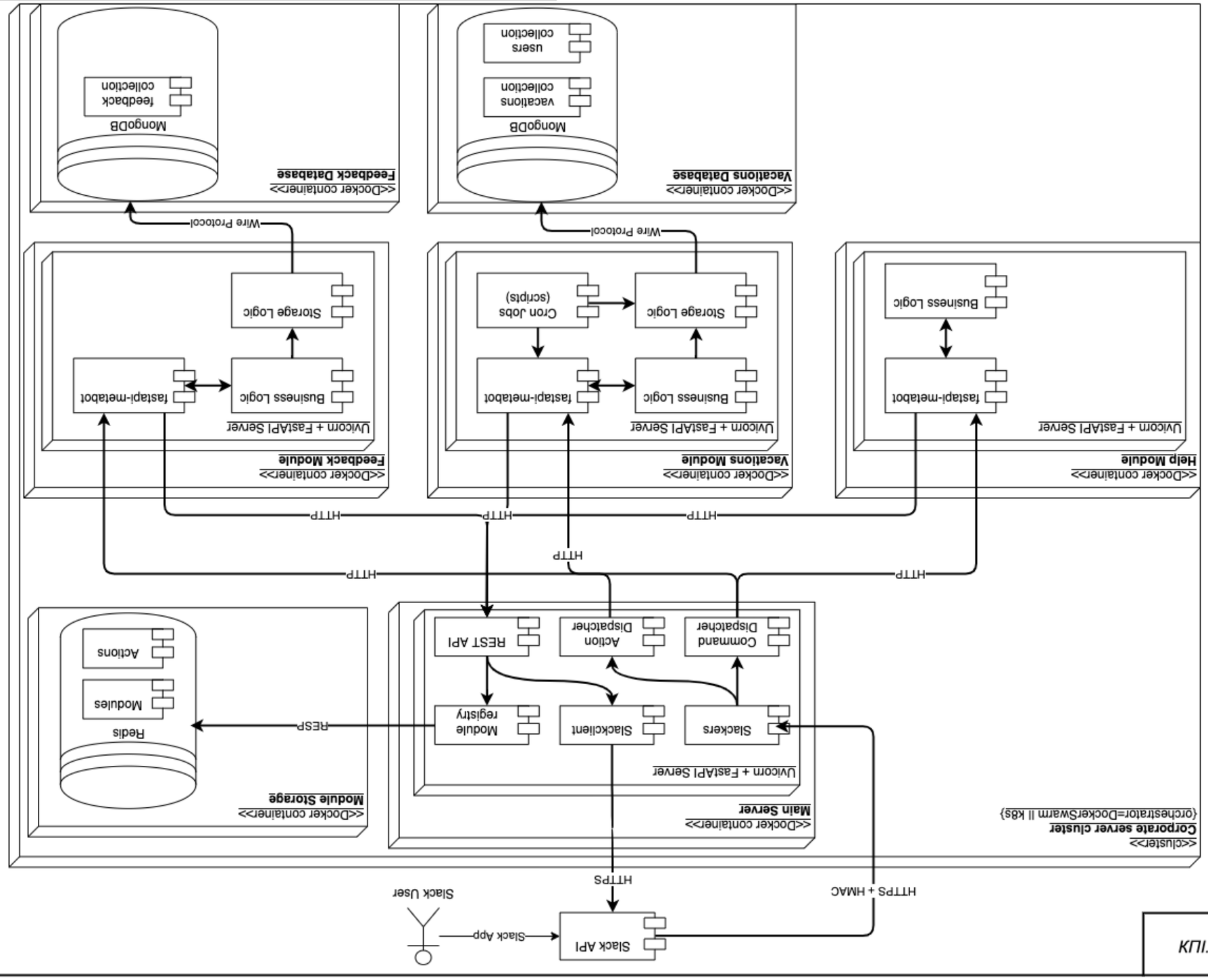
Рисунок 4 – Приклад додавання модуля в docker-compose.yml

Всі вбудовані модулі бота (help, vacations, feedback) також використовують бібліотеку, описану в даному керівництві, тож програмний код даних модулів можна використовувати в якості прикладу при розробці (наприклад, при організації файлової структури модуля, написанні Dockerfile тощо).

КПІ.ІП-6305.045490.07.99.СС

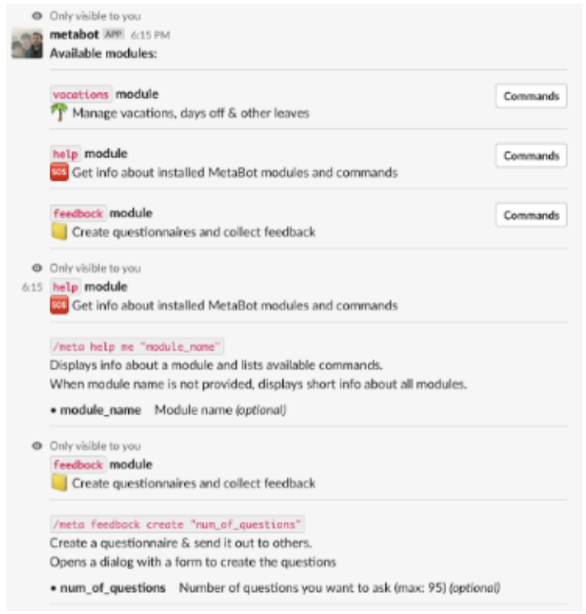


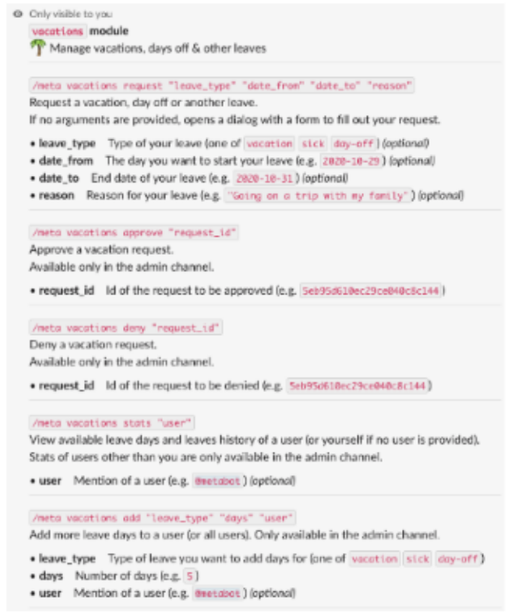
					КПІ.ІП-6305.045490.07.99.СС	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна бізнес процесів	Літера / Маса / Масштаб
Розробив		Васюк В.В.				
Перевірив		Коротенко А.А.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Корпоративний бот для ІТ-компаній	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Затвердив		Коротенко А.А.				

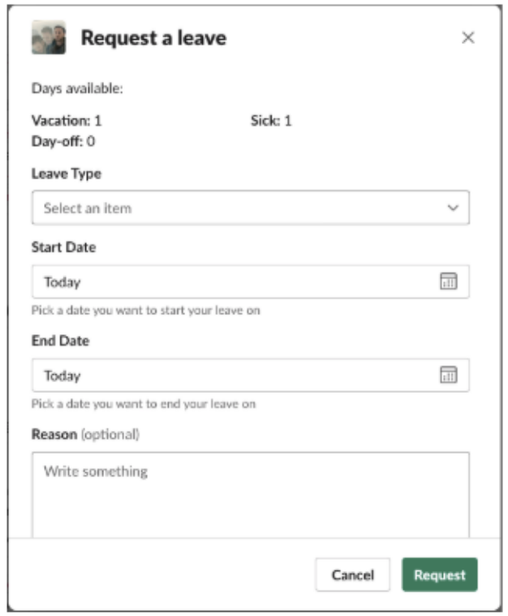


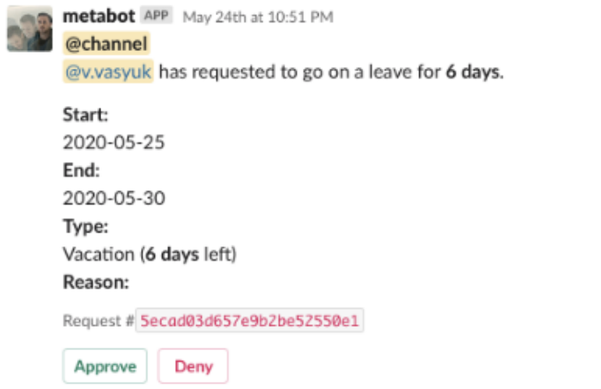
КПІ.ІП-6305.045490.08.99.СС

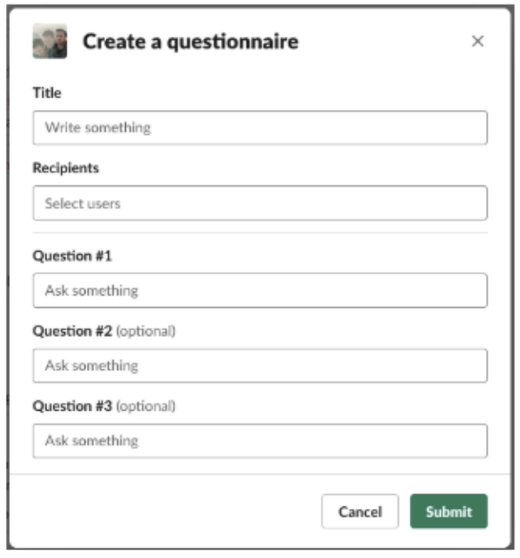
Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІП-6305.045490.08.99.СС		
Розробив		Бачок В.В.			Схема структурна розгортання	Літера	Маса	Масштаб
Перевірив		Коротенко А.А.						
Т. контр.						Аркуш 1	Аркушів 1
Н. контр.		Лілук К.І.			КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		
Затвердив		Коротенко А.А.			Корпоративний бот для IT-компаній		

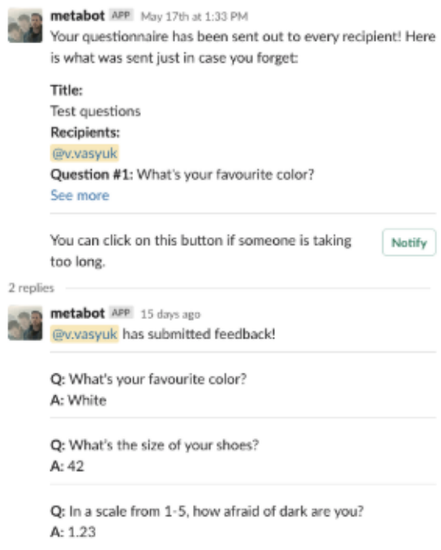


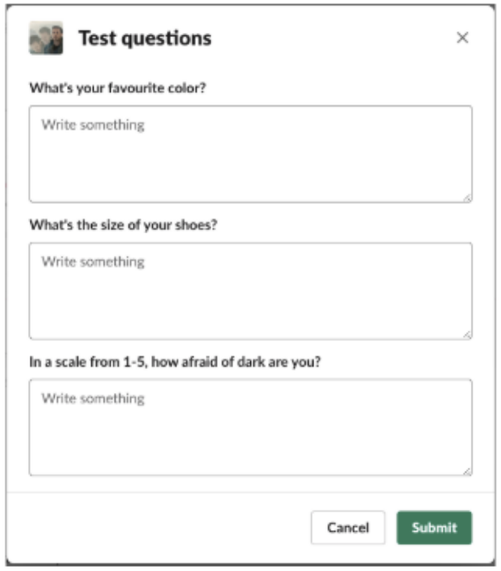


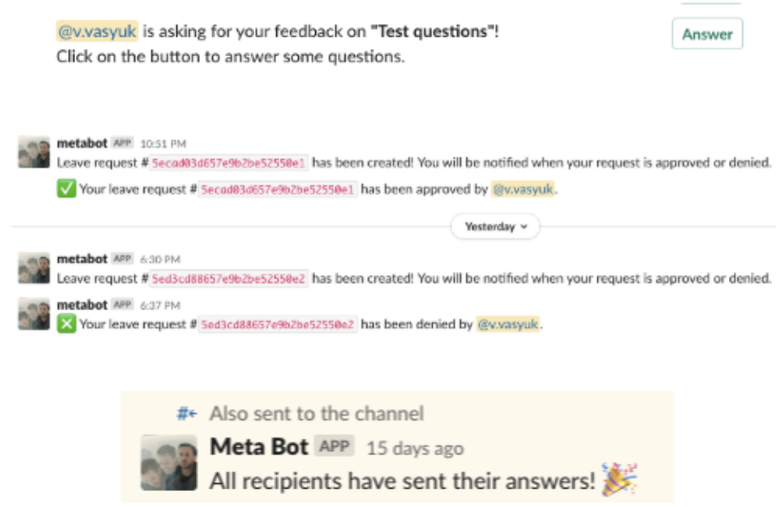












					КПІ.ІП-6305.045490.09.99.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Васюк В.В.			Креслення вигляду екранних форм	Літера / Маса / Масштаб
Перевірив		Коротенко А.А.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Корпоративний бот для ІТ-компаній	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Затвердив		Коротенко А.А.				